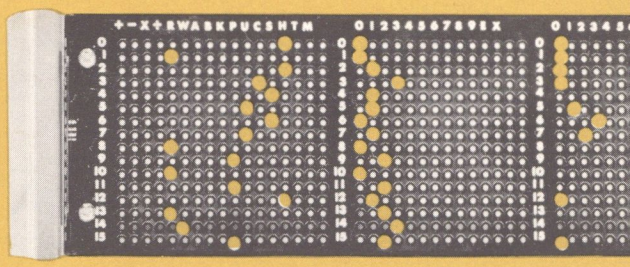
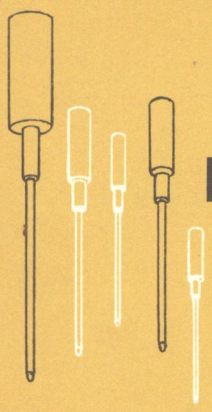
BULLETIN E175 A

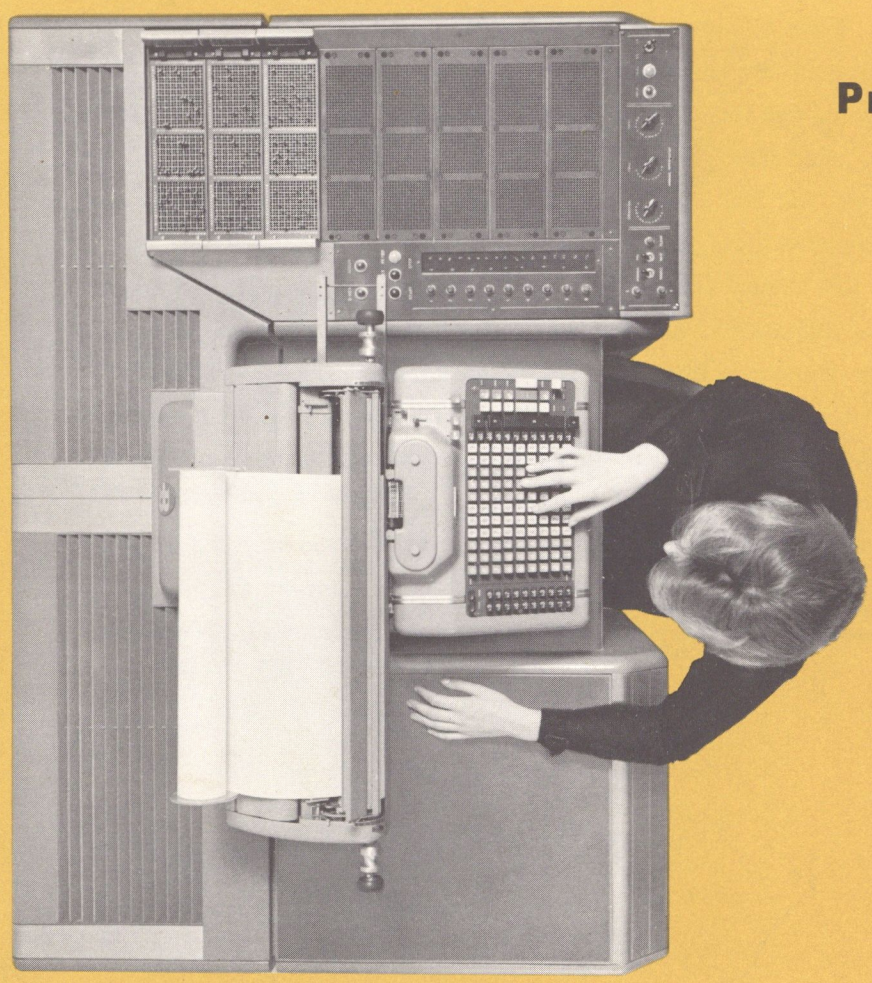




ElectroData 101

HANDBOOK

Advanced Programming



The ELECTRODATA 101 Desk Size Electronic Computer incorporates many of the features found in larger general purpose computers, and yet it is easy to program and operate. Many people with no prior computer experience have become skillful programmers after a few days' instruction.

This manual contains the information necessary to program the E101. It is used as a text in the Programming Courses conducted for E101 users.

Part I describes in detail the function of each command in the E101's instruction language, and tells how to operate the computer. Sample forms used in programming and operating are included.

Part II covers the use of the optional features which are available as adjuncts to the E101 (the 220-word memory, the tape input unit, etc.).

Part III is entitled "Programming Aids." It contains information on scaling, debugging, timing and checking circuitry. A list of some of the subroutines (available to all E101 users) is included.

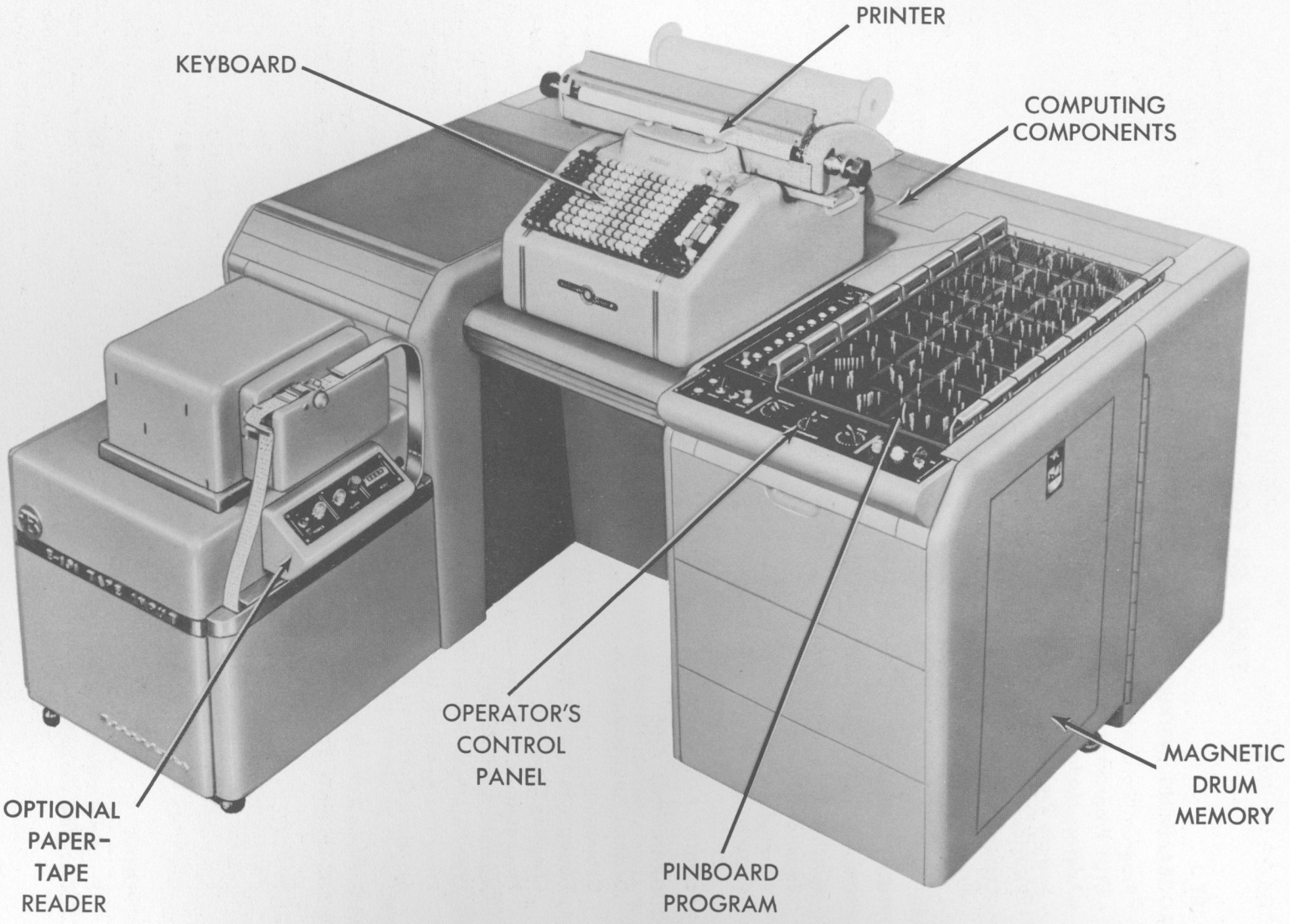
In Part IV you will find a discussion of programming strategy along with some valuable programming tricks or techniques.

Throughout this manual are sample programs to illustrate the text, and practice problems to test your grasp of material. In the Appendix are answers to the practice problems, and a summarized instruction list for quick reference.

Many E101 users contributed helpful suggestions for improving the Preliminary Edition of the Programming Manual. ELECTRODATA *wishes to thank all who thus assisted in producing this manual in its present form.*

If you have never programmed a digital computer before, you may find it helpful to read *You Can Program the* ELECTRODATA *101* before starting this manual. It explains in elementary terms just what programming is, and covers the rudiments of the E101.

TABLE OF CONTENTS



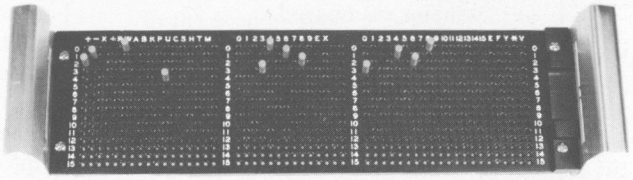
THE ELECTRODATA 101 ELECTRONIC COMPUTER

PART I

PROGRAMMING INSTRUCTIONS FOR THE BASIC E101

PROGRAM STORAGE IN THE E101

The program of the ELECTRODATA 101 is externally stored in removable pinboards. The machine instructions are selected by the positions of pins in the pinboards. There are positions for 8 pinboards numbered from 1 to 8, each with 16 steps numbered from 0 to 15.

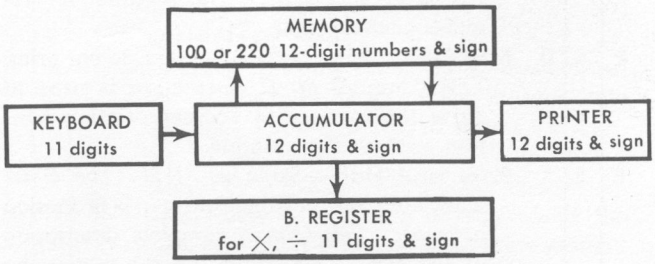


The illustration above shows a typical pinboard. Each instruction in the program is selected by the pins placed in a single horizontal row of holes of the pinboard. The first instruction (step 0) is W 4 8 which means, "Write the number now in the accumulator into memory address 48." Notice that there are three pins—one in each area of the pinboard. The pin in Area 1 selects the operation, while the pins in Areas 2 and 3 select the tens and units digits, respectively, of the memory address.

The instructions for a program are usually marked (with a pencil or ink stamp) on pre-punched paper templates; then the templates are laid over the pinboards and the pins dropped in place. This technique provides a permanent file copy of the program, which can be re-pinned in a few minutes, and virtually eliminates pinning errors.

E101 DATA FLOW

The E101 Data Flow Diagram, as the name implies, shows how data flows between the various parts of the computer. It is not meant to be a complete computer block diagram, as no mention is made of the pinboard program which controls the flow of data as well as all arithmetic operations. It is helpful to keep a picture of the Data Flow Diagram in mind when learning the programming instructions.



INPUT

The keyboard is the means of input on the basic E101. It is an 11-column full keyboard—the same type as used on automatic accounting machines or desk calculators. Punched tape may be used as an alternative means of input (a description of the characteristics of the tape input unit is included in Part II).

ACCUMULATOR

As indicated in the data flow diagram, the accumulator is the central clearing-house: all keyboard entries go into it, all results are printed from it, and the memory is loaded from it and read into it. Transfer of information from the accumulator to the memory, printer, or B Register leaves the number in the accumulator. It remains there until it is erased by an arithmetic operation or by another number coming into the accumulator from the keyboard or memory. The capacity of the accumulator is 12 decimal digits plus sign. The decimal point is located eleven places from the right (between the first and second digits on the left).

OUTPUT

The output unit of the E101 is a very flexible semi-ganged printer that prints the full 12-digit number in the accumulator in one operation. The flexibility of the printer enables reports to be prepared in almost any desired form. The printer is equipped with a carriage control panel that determines in what form reports will be printed. The control panel is set up in accordance with customers' specifications by an ELECTRODATA field engineer, making it unnecessary for the programmer to program format control. An optional tape punch provides an additional means of output. (A description of the tape output unit is included in Part II.)

MEMORY

The E101 has a magnetic drum memory with 100 memory locations numbered from 00 to 99; the capacity of each location is 12 decimal digits plus sign. The memory is used for the storage of numerical data only (the program is externally stored in the pinboards). Transfer of information from a memory address to the accumulator leaves the number in the memory, where it remains until it is erased by another number written into that address (from the accumulator, or as a result of a division). A 220-word memory is available as an optional feature (a description of the 220-word memory adjunct appears in Part II).

B REGISTER

The B Register is an additional data storage location, used for a multiplicand or a divisor. Once a number is transferred into the B Register, it remains there until it is erased by another number coming into the B Register from the accumulator.

NOTE: The B Register, like the keyboard, holds an 11-digit number while the memory, accumulator, and printer each hold 12-digit numbers. As will be explained later, the B Register is limited to 11 digits in order to prevent the loss of significant figures in multiplication. The keyboard is limited to 11 digits in accordance with the customary procedure of having one less digit in the keyboard than in the accumulator. *In both cases, the 11 digits correspond to the 11 least significant (right hand) digits of the accumulator, the printed number, or a memory location.*

E101 PROGRAMMING INSTRUCTIONS

This section contains detailed descriptive information on the programming instructions for the basic E101. It is generally helpful to refer to the data flow diagram on p. 1 when learning the instructions, especially K, W, R, B, and P.

Most instructions consist of three characters corresponding to the three areas of the pinboard. The address portion of the instruction is designated as "a b," "a" standing for the tens digit of the memory address and "b" for the units digit.

NOTE: Sometimes the programmer will mistakenly call for an instruction which generates an overflow. When this occurs, one of the E101's internal checking circuits halts the program and lights the ALARM signal. For a detailed description of the ALARM condition, and what the operator must do to continue computation, see the section on Checking Circuitry beginning on page 21.

PROGRAMMING INSTRUCTIONS FOR THE BASIC E101

PINBOARD AREA 1	2	3	OPERATION CARRIED OUT
K	—	—	**Keyboard**—Halt the program, and light the KEYBOARD signal. When one of the four motor bars is depressed, (a) transfer the number in the keyboard into the 11 least significant (right hand) digit positions in the accumulator, making the most significant (leftmost) digit zero; (b) if the minus bar is depressed, make the number negative; (c) print the number in the present carriage position; and (d) move the carriage to the next printing postition, depending upon which one of the four motor bars has been depressed.
K	—	0	**Keyboard Non-Print**—Same as "K" except do not print the number in the keyboard.
W	a	b	**Write**—Write the contents of the accumulator into memory address "a b" after first erasing old contents of "a b". Accumulator remains unchanged.
R	a	b	**Read**—Read the contents of memory address "a b" into the accumulator after first erasing old contents of accumulator. "a b" remains unchanged.
B	—	—	**B Transfer**—Transfer the contents of the accumulator into the B Register after first erasing old contents of B Register. Accumulator remains unchanged. NOTE: The capacity of the B Register is 11 digits. If the number transferring from the accumulator to the B Register exceeds 11 digits, i.e., exceeds 0.999 999 999 99, the E101 alarms. This is a safeguard built into the machine to prevent loss of significant digits in multiplication. In multiplication, the number in the B Register ($< 1.0 \times 10^0$) is multiplied by the number in a specified memory address (which can be as large as $9.999\ 999\ 999\ 99 \times 10^0$) leaving the answer in the accumulator which, like the memory, has a capacity for 12 digits or a number as large as $9.999\ 999\ 999\ 99 \times 10^0$. Since the number in the B Register is less than 1.0 in absolute value, the product cannot possibly exceed $9.999\ 999\ 999\ 99 \times 10^0$. If this safeguard were not built into the E101, the E101 might be called upon to multiply a number such as 4.000 000 000 00 in the B Register by a number such as 9.000 000 000 00 in the memory resulting in a product of 36.000 000 000 00. Since the accumulator has capacity for 12 digits with the decimal point 11 places from the right, the "3" in the answer would be lost.
P	a	—	**Print**—Print contents of the accumulator, using motor bar "a". The pin for "a" is set at 1, 2, 3, or 4, depending on the carriage motion desired after printing has occurred. If no pin is inserted for "a" in a P instruction, or if a number which is not 1, 3, or 4 is pinned for "a", the printer will automatically activate motor bar 2. (As in the case of a keyboard entry, the choice of motor bar depends on what format is desired. A fuller explanation of the motor bars is included in the section on Operating the E101.) The accumulator remains unchanged.
P	a	0	**Non-Print**—Same as "P a" except do not print. This "dummy print" instruction is used to move the carriage to another printing position, without first printing.
P	a	*	**Print and Halt**—Same as "P a". The E101 halts after the printing and carriage motion have occurred. (For a complete description of the HALT condition, see the section on Operating the E101.)

PINBOARD AREA 1	2	3	OPERATION CARRIED OUT
+	a	b	**Add**—Add the contents of memory location "a b" to the contents of the accumulator. The sum appears in the accumulator; the number in "a b" remains unchanged. NOTE: If the sum exceeds the 12-digit capacity of the accumulator (for example, if 6.000 000 000 00 in "a b" is added to 7.000 000 000 00 in the accumulator, resulting in a sum of 13.000 000 000 00), the E101 alarms, indicating that a "1" has been lost. The number 3.000 000 000 00 appears in the accumulator.
−	a	b	**Subtract**—Subtract the contents of "a b" from the contents of the accumulator. The remainder appears in the accumulator; the contents of "a b" remains unchanged. NOTE: Here again there is an alarm if the answer exceeds the 12-digit capacity of the accumulator. (For example if 6.000 000 000 00 is subtracted from −8.000 000 000 00 in the accumulator, the answer −14.000 000 000 00 would appear in the accumulator as 4.000 000 000 00,— and the alarm would sound, indicating an overflow.)
×	a	b	**Multiply**—Multiply the contents of the B Register by the contents of memory location "a b". The product appears in the accumulator; the contents of the B Register and of memory location "a b" remain unchanged. (As previously explained, in multiplication the number in the B Register must be less than 1.0×10^0 in absolute value. In order to prevent error, an alarm occurs if a "B" instruction calls for transfer of a number greater than or equal to 1.0 into the B Register. Multiplication is consequently an 11 digit by 12 digit operation. The most significant 12 digits of the product appear in the accumulator; round off of the product does *not* occur automatically; the least significant 11 digits are computed but not retained.)
÷	a	b	**Divide**—Divide the contents of the accumulator by the number in the B Register and store the quotient in memory location "a b". The undivided remainder appears in the accumulator one place to the left of the divisor; the contents of the B Register remain unchanged. (An alarm occurs if the absolute value of the quotient is $\geqq 10.0 \times 10^0$ resulting in the loss of significant digits.)
A	1	b	**Shift Left**—Shift the contents of the accumulator "b" places to the left, filling in zeros from the right. Pinning $0 \leqq b \leqq 10$ selects the number of places shifted. A 1 0, calling for a "shift" of 0 places, is a dummy instruction, used where a filler step is required. If the third area is left blank, or if any other number is pinned for "b", a shift of ten places will occur. (This is an off-end shift instruction used in extracting, scaling, positioning numbers prior to printing, and zeroizing the accumulator. When a number is shifted out of the accumulator by an A instruction, the loss of digits does not cause an alarm.)
A	2	b	**Shift Right**—Shift the contents of the accumulator "b" places to the right, filling in zeros from the left. (Like A 1 b, A 2 b is an off-end shift instruction, where an overflow to the right results in the loss of digits, but does not cause an alarm.)
A	3	−	**Absolute Value**—Make contents of the accumulator positive (regardless of former sign).
A	4	−	**Negative of Absolute Value**—Make the contents of the accumulator negative (regardless of former sign).
A	5	−	**Change Sign**—Change the sign of the contents of the accumulator.
A		*	**Halt**—Halt the machine. (This instruction is frequently used to allow the operator to override the program at strategic points in the problem. To resume operation, the operator depresses one of the nine START buttons on the control panel which are illustrated and explained in the section on "Operating the E101.")
U	a	b	**Unconditional Transfer**—Execute the instruction on pinboard "a", step "b". (The unconditional transfer is used most frequently to transfer back to the beginning of a loop of instructions in order to iterate within the loop— i.e., repeat the instructions. It is also the standard means of going from the last instruction in one pinboard to another pinboard. If the instruction in step 15 of a pinboard is not an unconditional transfer, the program will go to the first instruction—step 0—in the same pinboard.)
U	0	b	**Unconditional Transfer Within a Pinboard**—Execute instruction "b" on the same pinboard. (This instruction is helpful in the case of any one-pinboard routine—e.g., loading or clearing—that is used frequently. By pinning a "0" in the second area instead of a definite pinboard number, the operator can use the pinboard in any of the 8 pinboard positions *without* first changing this pin.)
U	a	*	**Unconditional Transfer to Next Step on Pinboard "a"**—Execute the instruction on pinboard "a" immediately following the last instruction previously executed in pinboard "a".
C	a	b	**Conditional Transfer**—Execute the instruction on pinboard "a", step "b", if the contents of the accumulator is negative; if not, continue with the next step in the program. (Zero is considered positive. If "a" and "b" are left blank, the "C−−" will function normally when

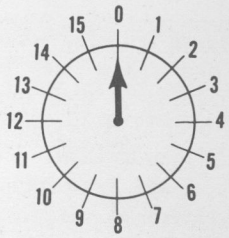
PINBOARD AREA 1	2	3	OPERATION CARRIED OUT
			the contents of the accumulator is positive, but will stop the program if the accumulator is negative.)
C	0	b	Same as "C a b" except stay in same pinboard on transfer.
C	a	*	Same as "C a b" except execute the instruction on pinboard "a" immediately following the last instruction previously executed in pinboard "a".
H	0	b	**Home E Switch**
H	1	b	**Home F Switch**
H	2	b	**Home E and F**
S	0	b	**Step E Switch**
S	1	b	**Step F Switch**
S	2	b	**Step E and F**

These are the automatic address modification instructions. Before proceeding with their description (which starts on p. 5), an explanation of the E and F switches is given.

ADDRESS MODIFICATION

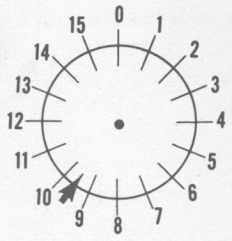
Automatic Address Modification

Automatic address modification is accomplished in the E101 by means of the E and F switches, which are two internal stepping switches. Basically, they are nothing more than counters. Each one is a 16-position switch that can count from 0 up through 15.

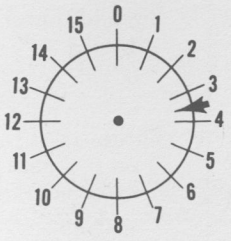


There are two programming instructions used to control each switch, one to set it at some initial position, and another to step it ahead one position at a time. To differentiate between the E and F switches in these instructions,we use the digit "0" to indicate the E switch and the digit "1" to indicate the F switch: E = 0, F = 1. Each switch can be set initially, or "homed," to any of its 16 positions by the "H" instruction, which is referred to as the "homing" instruction. "H 0 4" for example, homes the E switch to 4, while "H 1 6" homes the F switch to 6.

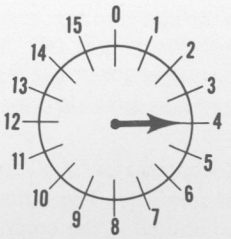
Each switch can be stepped ahead one position at a time and told at what position to stop its counting by the "S" instruction which is referred to as the "stepping" instruction. "S 0 9," for example, steps the E switch once each time the instruction is executed, stopping when E passes 9. "S 1 3" steps the F switch one position at a time stopping when F passes 3.



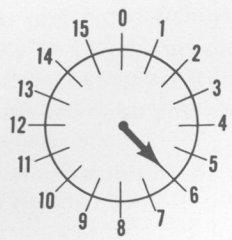
S09



S13



H04



H16

The "S" instruction stops the counting by means of a kind of conditional transfer. Recall that in the case of the unconditional transfer, "U," the E101 always goes to whatever program step is pinned along with the "U" (e.g., U 8 12). In the case of the conditional transfer, "C," the E101 automatically goes on with the next instruction in the program *unless* the number in the accumulator is negative. It transfers to the program step pinned along with the "C" (e.g., C 6 3) if, and only if, the accumulator is negative.

The "S" instruction is similar to the "C" in that it, too, is a conditional transfer. But the "S" instruction is more complex: first it advances the switch one position, *then* it looks at the pin in its third area to see if the switch has just passed the value pinned. As long as the switch has *not* just passed the limit pinned at "b," the program goes on to the next instruction in the pinboard. However, when the execution of the "S" instruction steps the switch past the limit which is pinned, the E101 *skips* the next instruction in the program and transfers instead to the instruction *after* the next instruction.

The "S" instruction is normally used in a block of steps which are repeated, called a "loop." The step after the "S" instruction is generally a "U" instruction which transfers back to the beginning of the loop to repeat the instructions (or to "iterate"). Each time through the loop, the "S" instruction steps the E or F switch (depending on which is called for) one position. The "U" instruction then takes the E101 back to the beginning of the loop to repeat the instructions. When E (or F) reaches the limit that is pinned (for example, when E *reaches* 9 in the case of the instruction "S 0 9"), the E101 *again* proceeds to the "U" instruction which takes the E101 back to repeat the loop. However, the next time the E101 comes to the "S" instruction, it steps E *from 9 to 10;* at this point it skips the "U" instruction and goes on with the rest of the program.

Although the E and F switches are sometimes used simply as counters, their most powerful feature is the fact

that the E101 *can use the numbers at which the switches are set as the address in an instruction.* In referring to the switch settings, we use the symbols "E" and "F." They can be used singly or in combination. The instruction "R E F," for example, tells the E101 to read into the accumulator the number which is stored in the memory location whose tens digit is to be found at the E switch setting, and whose units digit is the position of the F switch. The instruction "W 6 F" tells the E101 to write the contents of the accumulator into memory location 6 F, where F is the position at which the F switch is set. While E is generally used in the tens position and F in the units position, E can be used in either or both positions (e.g., W E F, − E 8, R 9 E, × E E).

The primary use of the E and F switches is to modify the addresses of instructions in a program automatically. The "H" instruction allows us to begin our address sequencing where we wish (usually at zero), and the "S" instruction enables us to advance through the sequence one step at a time. Although the E and F switches are 16-position counters, only the first ten positions (0 through 9) are used for automatic address modification. The next section, "Sample Programs," contains several illustrations of the use of the E and F switches in automatic address modification.

E and F, while generally used to denote an address in the memory, are not confined to this function. They may be used in the second and third areas of *any* instruction in place of specific numbers. The E101 will automatically substitute the setting of the switch for the symbol "E" or "F." As an example, the instruction, "U E *," is frequently used to return from a pinboard containing a subroutine to the main part of the program. Before transferring *to* the subroutine from one of several pinboards, the E switch is homed to the position corresponding to that pinboard number. The last instruction in the subroutine, "U E *," automatically returns the E101 to the next instruction on pinboard E, which is the pinboard from which the transfer *to* the subroutine was made. "C E *," can be used in a similar fashion.

MANUAL ADDRESS MODIFICATION

Manual address modification is accomplished in the E101 by means of the X and Y keys on the left side of the keyboard (illustrated in the section on Operating the E101). Pinning "X" for "a" or "Y" for "b" in any instruction (similar to pinning "E" or "F") sends the E101 to the appropriate *keyboard setting of the X or Y keys for that particular value of "a" or b."* One of the principal uses of the X and Y keys is to allow the operator random access to any location in the memory.

Just as in the case of the E and F switches, "X" and "Y" are used primarily for address modification but may be used with *any* instruction as variables in the second and third areas of the pinboard. Remember that there are ten X and ten Y keys: $00 \leqq XY \leqq 99$.

ADDRESS MODIFICATION INSTRUCTIONS

PINBOARD AREA 1	2	3	
H	0	b	**Home E Switch**—Home the E switch to position "b".
H	1	b	**Home F Switch**—Home the F switch to position "b".
H	2	b	**Home E and Move F**—Home the E switch to "b", advancing F in tandem with E. (This instruction advances E and F an equal number of steps but does not necessarily home them to the same location unless they were both at the same location initially.)
S	0	b	**Step E Switch**—Step the E switch once; then if the switch setting $E \neq b + 1$, execute the next instruction; if $E = b + 1$, skip the next instruction, and instead execute the instruction *after* the next instruction. (Since E is a 16-position switch, if stepped while in position 15, it will go to its position 0. If b is left blank, the S instruction will step the switch, but will never skip the next instruction; that is, in the absence of a "b" pin, it will always execute the next instruction.)
S	1	b	**Step F Switch**—Same, but using F switch.
S	2	b	**Step E and F**—Same using E and F switches. Skip the next instruction when E passes "b".
_	E	F	Execute the instruction pinned in the first area (for example, W, R, +, −), using the E switch setting as the tens digit and the F switch setting as the units digit of the address.
_	E	b	Execute the instruction pinned in the first area, using the E switch setting as the tens digit of the address.
_	a	E	Execute the instruction pinned in the first area, using the E switch setting as the units digit of the address.
_	E	E	Execute the instruction pinned in the first area, using the E switch setting as both the tens and units digits of the address.
_	a	F	Execute the instruction pinned in the first area, using the F switch setting as the units digit of the address.
_	X	Y	Execute the instruction pinned in the first area (for example, W, R, +, −), using the keyboard setting for X as the tens digit, and the keyboard setting for Y as the units digit of the address.
_	X	b	Execute the instruction pinned in the first area, using the keyboard setting for X as the tens digit of the address.
_	a	Y	Execute the instruction pinned in the first area, using the keyboard setting for Y as the units digit of the address.
U	E	*	Execute the instruction on pinboard E (corresponding to the E switch setting) immediately following the last instruction previously executed there.
C	E	*	Execute the instruction on pinboard E (corresponding to the E switch setting) immediately following the last instruction previously executed there if the contents of the accumulator

is negative. If not, go to the next instruction in the program.

SAMPLE PROGRAMS

This section contains sample programs which the reader can use as a means of reviewing the programming instructions given in the preceding section. We have included a few "homework" problems at the end for anyone who wants additional practice. The "answers" can be found in the Appendix.

1) $y = ax^2 + b$

0	K	Enter a in keyboard
1	W 0 0	Write a into memory location 00
2	K	Enter b in keyboard
3	W 0 1	Write b into memory location 01
4	K	Enter x in keyboard
5	B	Store x in B Register
6	× 0 0	Multiply a by x
7	W 0 2	Write ax into memory location 02
8	× 0 2	Multiply ax by x
9	+ 0 1	Add b to ax^2
10	P 1	Print $ax^2 + b$ (using motor bar 1)
11	U 0 4	Transfer back to step 4 of this pinboard

2) $y = x - \frac{a}{x^2}$

0	K	Enter a in keyboard
1	W 0 0	Write a into memory location 00
2	K	Enter x in keyboard
3	W 0 1	Write x into memory location 01
4	B	Store x in B Register
5	× 0 1	Multiply x by x
6	B	Store x^2 in B Register
7	R 0 0	Read a out of memory location 00 into accumulator
8	÷ 0 2	Divide a by x^2; store quotient in memory location 02
9	R 0 1	Read x out of memory location 01 into accumulator
10	− 0 2	Subtract $\frac{a}{x^2}$ from x
11	P 1	Print $x - \frac{a}{x^2}$ (using motor bar 1)
12	U 0 2	Transfer back to step 2

3) Clear (i.e., erase contents of) memory locations 14, 16, 23 and 39:

0	R 9 9	Read contents of memory location 99 into accumulator
1	− 9 9	Subtract contents of memory location 99 from accumulator (leaving zero).
2	W 1 4	Write contents of accumulator (zero) into memory location 14
3	W 1 6	Write contents of accumulator into memory location 16
4	W 2 3	Write contents of accumulator into memory location 23
5	W 3 9	Write contents of accumulator into memory location 39
6	- - -	NOTE: The choice of location 99 in steps 0 and 1 is arbitrary: *any* location will do as well.

Another way of programming this problem is as follows:

0	A 2 6	Shift contents of accumulator 6 places to the right (leaving zeros in the 6 digit positions at the left).
1	A 2 6	Shift contents of accumulator 6 places to the right (leaving zeros in all 12 digit positions).
2	W 1 4	Same as above
3	W 1 6	
4	W 2 3	
5	W 3 9	
6	- - -	

Steps 0 and 1 could just as easily have been A 1 6 and A 1 6 (shift left). Still another way of programming this problem is as follows:

0	K	Enter zero in keyboard. (This is done by touching a motor bar without indexing a number on the keyboard.)
1	W 1 4	Same as above
2	W 1 6	
3	W 2 3	
4	W 3 9	
5	- - -	

This method uses one less step than the others, but requires operator participation.

4) Print contents of memory locations 40 to 49 without using the E or F switch:

Pinboard #1

0	R 4 0	Read contents of memory location 40 into accumulator.
1	P 2	Print contents of accumulator using motor bar 2.
2	R 4 1	Read contents of 41 into accumulator.
3	P 2	Print.
4	R 4 2	Read contents of 42 into accumulator.
5	P 2	Print.
6	R 4 3	Read contents of 43 into accumulator.
7	P 2	Print.
8	R 4 4	Read contents of 44 into accumulator.
9	P 2	Print.
10	R 4 5	Read contents of 45 into accumulator.
11	P 2	Print.
12	R 4 6	Read contents of 46 into accumulator.
13	P 2	Print.
14	R 4 7	Read contents of 47 into accumulator.
15	U 2 0	Transfer to pinboard 2, step 0.

Pinboard #2

0	P 2	Print.
1	R 4 8	Read contents of 48 into accumulator.
2	P 2	Print.
3	R 4 9	Read contents of 49 into accumulator.
4	P 2	Print.
5	- - -	

(Total of 21 steps.)

5) Print contents of memory locations 40 to 49 using F switch.

0	H 1 0	Home the F switch to zero.
→1	R 4 F	Read out contents of memory location 4F where F is the setting on the F switch. (The first time through the routine F will be 0; the next time it will be 1, then 2, etc.)
2	P 2	Print the contents of accumulator using motor bar 2.
3	S 1 9	Step the F switch. As long as F doesn't pass 9, go to the next instruction; when F passes 9, skip the next instruction and go on to instruction 5. (The first time through the routine S 1 9 will step F from 0 to 1; the next time from 1 to 2, etc.)
└4	U 0 1	Transfer back to step 1.
5	- - -	

(Total of only 5 steps compared to 21 steps in previous routine.)

6) Load 50 constants into memory locations 00 to 49:

0	H 0 0	Home the E switch to zero.
→1	H 1 0	Home the F switch to zero.
→2	K	Enter constant in keyboard.
3	W E F	Write into memory location EF (where E = setting of E switch and F = setting of F switch).
4	S 1 9	Step the F switch; if F ≠ 10, go to next instruction. When F = 10, skip the next instruction and go to step 6. (Each time the E101 reaches the S 1 9 instruction, F is increased by 1. Notice that E does not change as long as the program remains in the F loop. The first time through the routine the constant will be written into 00; the next time in 01, then in 02, etc. After the 10th constant has been written into 09, the E101 will skip the U instruction in step 5 and go on to step 6.)
└5	U 0 2	Transfer back to step 2.
6	S 0 4	Step the E switch once each time the E101 comes to this instruction (once every 10 keyboard entries). After E passes 4, skip to instruction 8.
└7	U 0 1	Transfer back to step 1 where F switch is again homed to zero. (E is homed only at the beginning.)
8	- - -	

PRACTICE PROBLEMS

1. Load 10 constants into memory locations 70 to 79 as follows:
   a) without using E or F switch
   b) using F switch
   c) using E switch
2. Enter a constant through the keyboard and store it in the B Register. Enter 10 numbers through the keyboard and store them in memory locations 00 to 09. Divide each one by the constant stored in the B Register and store the quotients in memory locations 20 to 29. Do this problem two ways:
   a) without using E or F switch
   b) using F switch
3. Enter 10 numbers through the keyboard and store them in memory locations 80 to 89. As they are entered, accumulate their total in memory location 99. Print out the final total.
4. Clear memory locations 00 to 39.

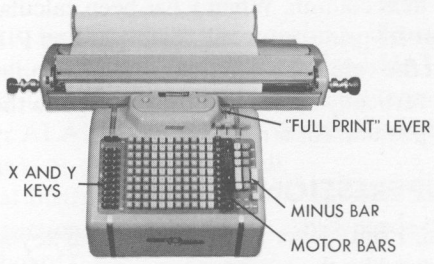
Answers

There are sometimes several ways in which a problem can be programmed. One method suggested for programming each one of the above problems can be found in the Appendix at the back of the Manual.

OPERATING THE E101

Anyone who has operated a desk calculator or accounting machine can be taught to operate the E101 with just a few hours of instruction. In some E101 installations, the computer is operated by personnel with little or no programming experience. A manual entitled *Handbook of Operating Instructions for the* ELECTRODATA *101* explains in detail how to run problems off on the machine, and yet assumes no knowledge of programming.

But although the E101 programmer need never become an expert operator, he must be able to *debug his own programs, and to specify clearly to others how his problems must be run.* The more he knows about the E101's input, manual control, and output features, the better will he be able to save program steps and minimize running time. For these reasons, a thorough reading of *Handbook of Operating Instructions for the* ELECTRODATA *101* is recommended. For the purposes of this Manual, however, the following remarks will suffice.



The E101 stops for a keyboard entry at each "K" instruction in the program. The operator enters the number through the 11 column full keyboard as if he were operating an adding machine. If the number is negative, he depresses the minus bar at the extreme right of the keyboard. To complete the operation, he touches one of the four motor bars located on the right of the 11 digits. Touching the motor bar does three things: puts the number into the E101 accumulator, prints it on the report (in red if negative), and moves the carriage to the next printing position. The position to which the carriage moves depends on which motor bar is touched.

The operator must use a motor bar every time the E101 calls for a keyboard entry. The programmer must not only specify to the operator which motor bars he is to use, but must also *tell the E101* which motor bar to employ each time it prints out a number from its accumulator.

When a motor bar is activated, the E101 prints. In general, the carriage motions which occur after printing are as follows:

MOTOR BAR	CARRIAGE MOTION
1	space vertically and return to left (to column 1)
2	move to next tab stop to the right
3	space vertically
4	tab to the right, skipping one or more columns

These basic functions can be altered to meet the varying format requirements of each problem: one motor bar can produce the carriage motion typical of another, the carriage can be made to move to the left, and so forth. An interchangeable control panel, which fits into the moving carriage, is set mechanically to call for four different columnar arrangements, and to override the basic motor bar functions in those columns where needed. A general purpose control panel is supplied with each E101; it can be changed, or additional control panels can be kept on hand for unlimited format flexibility.

As an example of how the motor bars are used, consider a problem in which the operator indexes values of x through the keyboard and the E101 computes the corresponding values of y and prints them next to the values of x: we want to end up with two vertical columns, "x" and "y." The operator would touch motor bar 2 after entering each value of x, thereby moving the carriage horizontally to the next column. When y has been calculated, the struction for its print-out would be pinned as P 1 _; this would print the value of y next to the corresponding value of x, space vertically and return the carriage to the left so that it is in position for a new value of x.

ZERO SUPPRESSION

The "Full Print" lever at the back of the keyboard on the right is used by the operator to suppress zeros in keyboards entries and printed results when desired. As long as the lever is in the forward position (marked "Full Print") the complete 12-digit number in the accumulator prints on the report (e.g., 0 000 000 638 20). When the operator moves the lever to the rear position (away from the operator), all zeros to the left of the first significant digit are suppressed (e.g., 638 20).

X AND Y KEYS

The small dark keys in the first two columns on the left of the keyboard are the X and Y keys used primarily for manual address modification. When X and Y are pinned in a program instead of a definite memory address, the operator determines which memory address the E101 should use by depressing a key in the X column (at the extreme left) and one in the Y column. The E101 uses these values in the program wherever X and Y are called for until the operator sets up new values in the X and Y keys—usually at the time of a keyboard entry or halt instruction. The X and Y keys remain set through all keyboard entries and printings; they can be cleared only from their own individual clear keys at the top of each column.

START BUTTONS

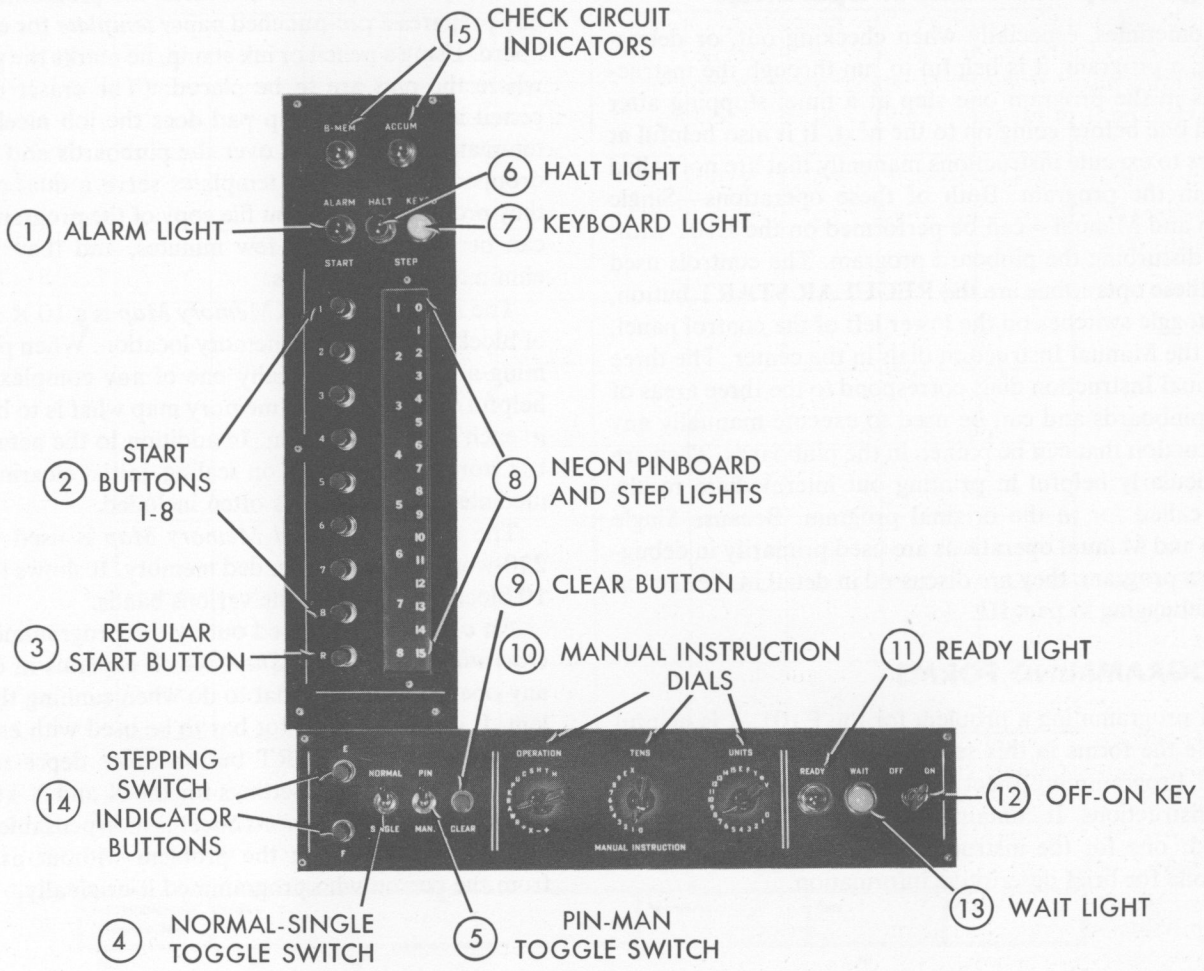
The E101 has been designed so that computation can be stopped at any point to allow human judgment to be brought to bear on the problem where required. At such points, the operator can decide what part of the problem the E101 should solve next, enter new numbers, etc. Trial-and-error problems are especially well-suited to the E101 because of this feature.

When programming a problem of this nature, the programmer calls for halt instructions (A _ * or P a *) at strategic points in the problem. When the E101 reaches a halt instruction, the program stops while the operator examines the results up to that point and decides what action to take. To resume operation, the operator touches one of the nine START buttons on the left of the neon lights. The first eight buttons are numbered from 1 to 8 and correspond to the eight pinboards. Touching any one of these buttons sends the E101 to the first instruction on the corresponding pinboard for its next instruction. The 9th START button, marked "R," is called the REGULAR START button; it sends the E101 to the next instruction in the program following the halt instruction.

As an example of how this feature may be used, an engineer might be running a trial-and-error design problem involving a number of design parameters. By programming a halt instruction at the proper place, he can examine the results at that point and decide whether to go on with the computation (in which case he would probably touch the REGULAR START button), whether to go back to the beginning of the problem and try a new parameter (in which case he might touch START button 1) or whether to skip ahead to the final print-out instructions on the last pinboard (in which case he would touch START button 8).

E101 CONTROL PANEL

1. ALARM light—lights up when E101 stops under certain conditions such as accumulator overflow.
2. START buttons 1 to 8—send E101 to first instruction on pinboards 1 to 8 respectively.
3. REGULAR START button:
   under PINBOARD control, NORMAL operation, it restarts the program at the next pinboard step;
   under PINBOARD control, SINGLE operation,



it executes the next pinboard step;
under MANUAL control, it executes whatever instruction has been set up in the Manual Instruction dial switches.

4. NORMAL/SINGLE toggle switch—in NORMAL position, E101 executes each pinboard step in sequence automatically; in SINGLE position, E101 executes one pinboard instruction at a time; used to stop automatic computation.
5. PIN/MAN toggle switch—In PIN position, E101 takes its instructions from the pinboards; in MAN position, E101 executes the instructions set up in Manual Instruction dials.
6. HALT light—lights up when E101 stops at a halt instruction or an alarm.
7. KEYBOARD light—lights up when E101 stops at a keyboard instruction.
8. Neon pinboard and step lights—Indicate pinboard number (left column) and step number (right column) of instruction being followed.
9. CLEAR button—halts the machine, clears the instruction flip-flop, and turns on the HALT light; used to allow operator to re-start computation after ALARM (it turns off the ALARM light and gong), or machine standstill.
10. Manual Instruction dials—allow operator to execute instructions manually, (i.e., independent of the pinboard).
11. READY light—indicates E101 is ready for operation.
12. OFF-ON key—turns E101 "on" and "off."
13. WAIT light—indicates E101 is "on," but is warming up.
14. STEPPING SWITCH INDICATOR BUTTONS—permit operator to read E or F switch positions from neon step lights *when machine has momentarily halted;* depressing both E and F buttons together reads out band switch setting.
15. CHECK CIRCUIT INDICATORS—light up when circuit malfunction occurs.

Single Step and Manual Operation

Sometimes, especially when checking out, or debugging, a program, it is helpful to run through the instructions in the program one step at a time, stopping after each one before going on to the next. It is also helpful at times to execute instructions manually that are not called for in the program. Both of these operations—Single Step and Manual—can be performed on the E101 without disturbing the pinboard program. The controls used for these operations are the REGULAR START button, the toggle switches on the lower left of the control panel, and the Manual Instruction dials in the center. The three Manual Instruction dials correspond to the three areas of the pinboards and can be used to execute manually any instruction that can be pinned in the pinboards. They are particularly helpful in printing out intermediate results not called for in the original program. Because Single Step and Manual operations are used primarily in debugging a program, they are discussed in detail in the section on debugging in part III.

PROGRAMMING FORMS

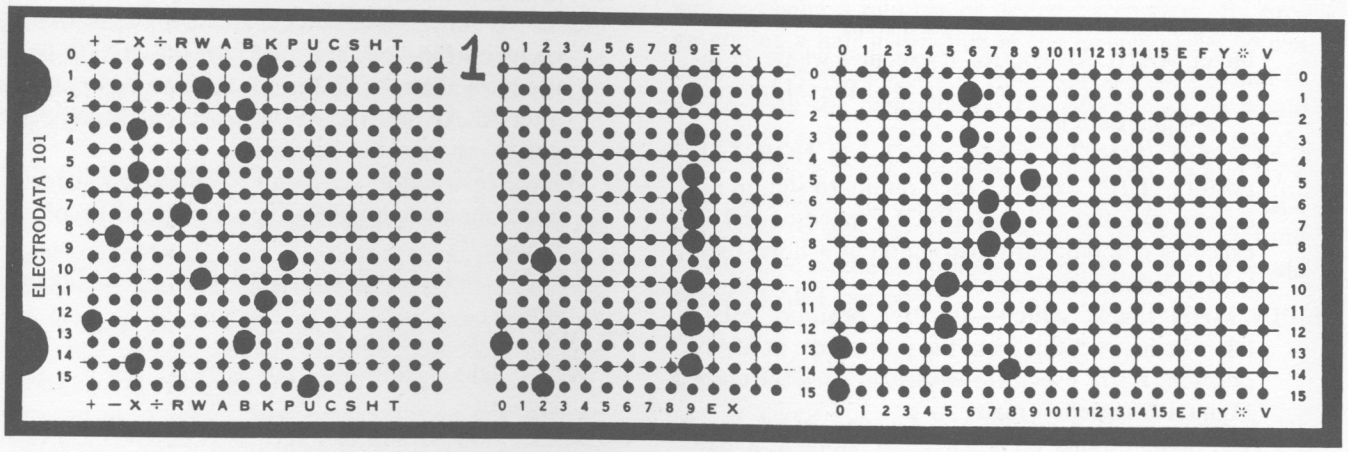
In programming a problem for the E101, it is helpful to use the forms in this section. The first of these is the E101 Programming Sheet on which the programmer lists the instructions. It contains three columns for each pinboard: one for the instructions, one for scaling factor, and one for brief descriptive information.

When the program is complete, the programmer usually prepares a pre-punched paper *template* for each pinboard. Using a pencil or ink stamp, he marks the positions where the pins are to be placed. (The eraser end of a pencil inked on a stamp pad does the job nicely.) The templates are then laid over the pinboards and the pins dropped in place. The templates serve a dual purpose: they provide a permanent file copy of the program, which can be re-pinned in a few minutes, and they virtually eliminate pinning errors.

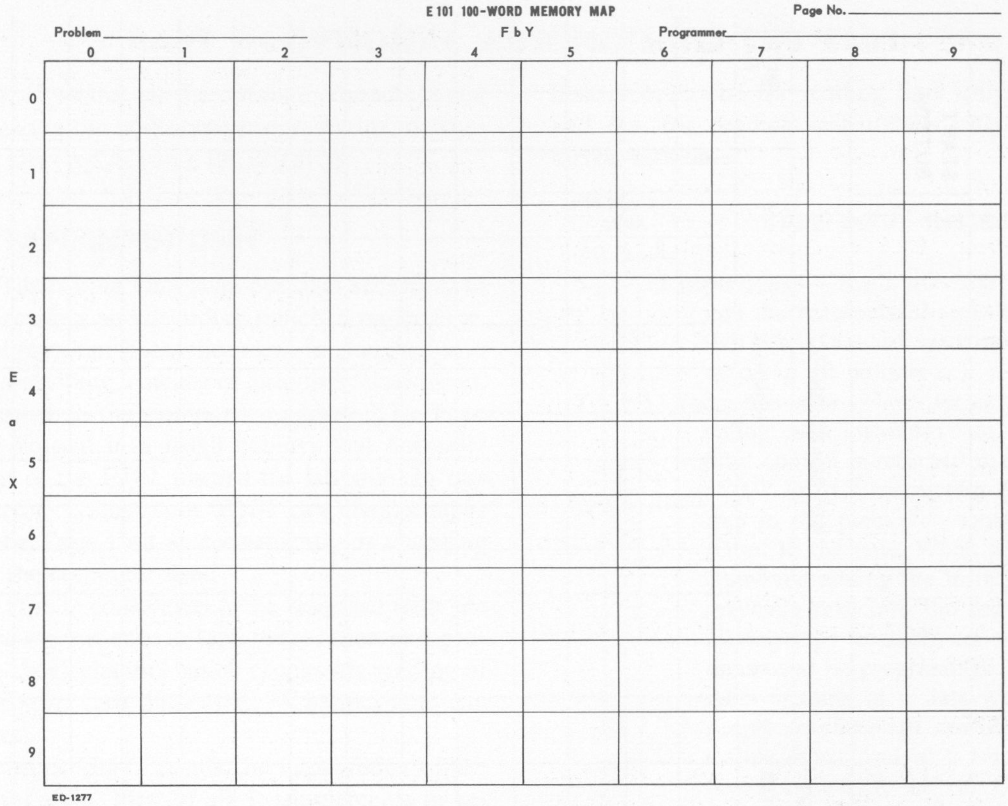
The *E101 100-Word Memory Map* is a 10 × 10 array of blocks, one for each memory location. When programming a problem, especially one of any complexity, it is helpful to record on the memory map what is to be stored in each memory location. In addition to the actual number stored, information on scaling, initial clearing, intermediate changes, etc., is often included.

The *E101 220-Word Memory Map* is used with the 220-word optional expanded memory. It shows the 22 × 10 blocks divided into the various bands.

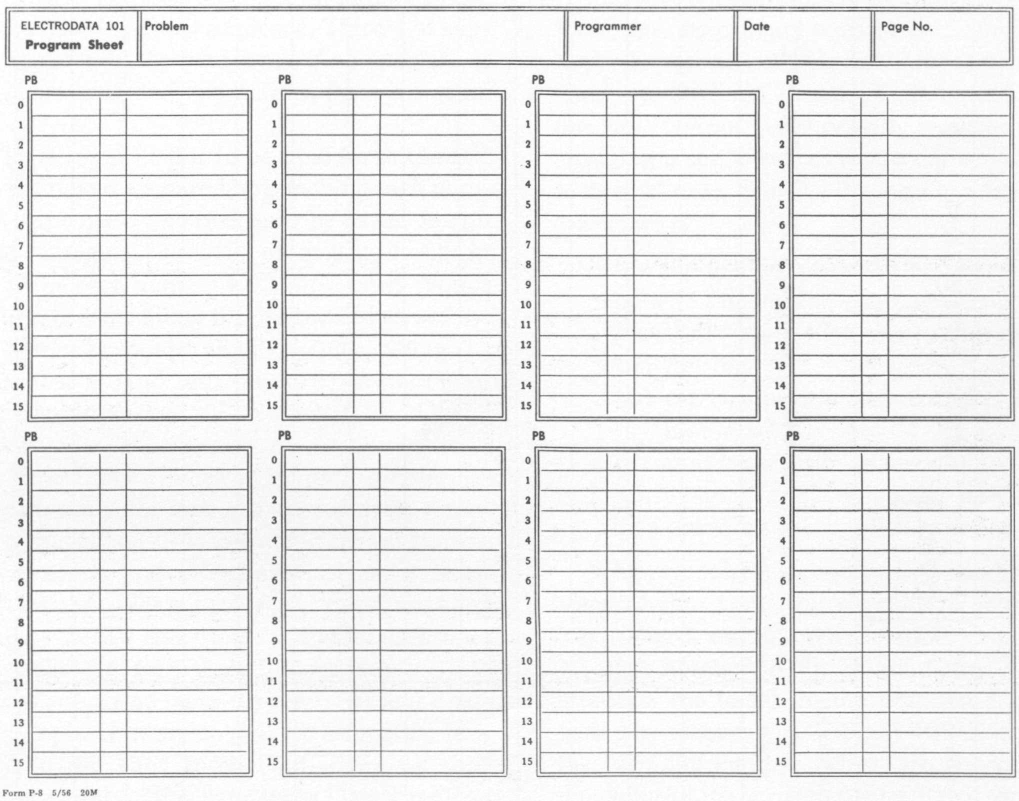
An essential form filled out by the programmer is the *Operating Instructions Sheet* which explains in detail to any operator exactly what to do when running the problem. It includes the motor bar to be used with each keyboard entry, the START button to be depressed after each halt instruction, settings for the X and Y keys, etc. The *Operating Instructions Sheet* is indispensable to anyone attempting to run the problem without assistance from the person who programmed it originally.



Template


E101 100-WORD MEMORY MAP
Page No.
Problem
F b Y
Programmer
0 1 2 3 4 5 6 7 8 9
E a X
0 1 2 3 4 5 6 7 8 9
ED-1277

E101 100-Word Memory Map


ELECTRODATA 101
Program Sheet
Problem
Programmer
Date
Page No.
PB
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Form P-8 5/56 20M

E101 Programming Sheet

OPERATING INSTRUCTIONS SHEET

ELECTRODATA 101

PROBLEM				PROGRAMMED BY	
SET-UP	PINBOARDS	PRINTING SCHEDULE	SPECIAL INSTRUCTIONS		START BUTTON

SIGNAL (K, H)	PINBOARD AND STEP		PINBOARD CHANGES, MANUAL INSTRUCTIONS, ETC.	START BUTTON	KEYBOARD ENTRY														MOTOR BAR
					X	Y	NUMBER												

PART II - OPTIONAL ADJUNCTS TO THE E101

Here in Part II we discuss the optional adjuncts to the E101, emphasizing the additional programming instructions that are involved in using the adjuncts.

PUNCHED TAPE INPUT UNIT

The E101 Tape Input Unit is an optional accessory to the E101 for reading and decoding punched paper tape. Both data and program instructions can be punched into the tape, thus providing a means of automatic data input, and a supplement to the pinboard program. The Tape Input Unit is housed in a small cabinet that normally stands in front of the E101 toward the left side. In this position a person sitting at the E101 can change tapes and operate the Tape Unit as conveniently as changing pinboards on the computer itself.

8-Channel tape is considered to be standard with the E101. It offers convenience in tape preparation and provides a parity-check channel which checks the reading of each tape character into the E101. 5-Channel tape can also be handled.

The Tape Input Unit extends the capabilities of the E101 in several ways. First of all, it supplements or replaces *keyboard entry* by providing an automatic means of loading constants into the memory and entering variable data. Secondly, it augments the capacity of the *memory* by making data available in those situations where constants can be used in a fixed sequence. Thirdly, it supplements the capacity of the *pinboards*. And fourthly, it supplies data, constants and instructions from the *same* tape.

When the Tape Input Unit is to be used for automatic entry of data, the tape is prepared by punching each number on tape in sequence. A number may be of any length between 1 and 12 digits. The E101 automatically fills in with zeros to make each number a 12-digit word. Notice that the tape unit permits filling the entire 12 digits of the accumulator while keyboard input permits filling 11 digits. The number is read into the E101 accumulator with the least significant digit all the way to the right of the accumulator.

When the Tape Input Unit is to be used for supplementary program steps, the tape is prepared using essentially the same instruction code as used in the pinboards. The only difference is that every tape instruction must consist of 3 characters. Where necessary, "ones" are filled in (for example, "K11" and "B11"). Instruction tapes are frequently spliced to form a continuous loop.

The details of preparing tapes for the E101 and operating the Tape Input Unit are covered in a separate brochure devoted to the Tape Input Unit—here we shall cover only those aspects of the Tape Input Unit that affect programming.

In programming a problem that involves tape input, the programmer has available two additional pinboard instructions: one for reading data only into the E101, and one for reading instructions only or instructions mixed with data.

TAPE INPUT INSTRUCTIONS

PINBOARD AREA 1	2	3	
T	—	12	**Tape Read**—Read the next number on tape into the accumulator and then continue with the next pinboard instruction. (Control remains in the pinboards.)
T	—	11	**Tape Transfer**—Transfer control to tape, executing each instruction on tape in sequence until control is returned to the pinboards by a "U" or "C" instruction on tape. (At any step in the pinboards control can be transferred to the tape. When a data word is encountered among the instructions, it is automatically read into the accumulator without leaving tape control; no additional instructions need be programmed for this to occur. After reading in a data word, the program simply continues to execute the instructions on the tape. Control is returned to the pinboards only by a "U" or "C" transfer on the tape, to any step in any pinboard.)

PUNCHED TAPE OUTPUT UNIT

The E101 Tape Output Unit is a new adjunct to the E101 line. Its primary purpose is to allow the E101 user to produce punched tape with *data or instructions* for reentry into the E101 (via the Tape Input Unit). Although the Tape Output Unit normally punches 8-channel tape, changing a single encoder card converts it for punching 5-channel tape. This flexible unit can be used in the following ways:

1. it will punch into tape the number in the accummulator (12 digits with sign), or the number in the keyboard (11 digits with sign), with or without
   (a) printing,
   (b) carriage positioning controlled by any of the 4 motor bars,
   (c) zero suppression to the left;
2. it will punch instruction codes into tape as specified by a pinboard or manual instruction.

When punching data words, including the sign, the Tape Output Unit automatically punches the necessary Begin Word, and End Word symbols. Normally the entire 12 digit number in the accumulator will be punched, however, the length of the data word punched can be varied by means of a ZERO SUPPRESS CONTROL on the Tape Output Unit. When ON, the control will inhibit punching all the zeros to the LEFT of the first significant digit of the number in the accumulator. This control operates independently of the FULL PRINT control on the Keyboard-Printer.

In using the Punched Tape Input Unit, the programmer has available five instructions for punching data from the accumulator, two instructions for punching the number which is in the keyboard, and one instruction for punching instruction codes into the tape. These instructions are listed in the Appendix on page 37.

EXPANDED MEMORY

The expanded memory adjunct to the E101 provides for 220 words of data storage instead of the standard 100 (each number consisting of 12 decimal digits plus sign).

The 220-word memory may be thought of as consisting of 5 bands on the drum: one permanent or "heart" band of 60 memory locations, and four "switchable" bands, each of 40 memory locations, which are selected by the program as needed. The permanent band is always available for use at any point in the program; only one of the four switchable bands, however, can be selected for use at one time. This may be compared to two storage cabinets standing side by side, the first with one large compartment and the second with four separate drawers. Since there are no drawers to open or close in the first cabinet, all parts of it can be reached at one time. In the second cabinet, however, only one drawer is accessible at any one time. All four drawers, of course, can be used for storage, but material can be placed in or removed from only one drawer at a time.

The four 40-word bands are designated 0, 1, 2 and 3. Each one contains memory addresses 00 to 39. Data, therefore, can be stored in memory address 23, band 0, or in memory address 23, band 1, band 2 or band 3. The memory locations in the heart band are numbered from 40 to 99 without reference to band number.

When programming a problem for the expanded memory E101, the programmer uses the same instructions as for the 100-word memory E101: "W a b," R a b," "+ a b," etc. When memory addresses 40 to 99 are called for, the E101 *automatically uses the heart band.* When 00 to 39 are called for, it uses the band corresponding to the setting of the *band switch.* This is an automatic switch, called the M switch, similar to the E and F switches. It can be set initially by the instruction "H 3 b" which homes M to position "b" (0, 1, 2 or 3) and can be stepped automatically (in the same manner as E and F) by the instruction "S 3 b," where "b" is the upper limit.

> As long as a problem requires no more than 100 memory locations, there is no need to select one of the four bands; the band selector switch is always set at one of its four positions, and which band is being used is irrelevant *as long as the program does not call for a change in the band switch setting.* It follows that any program which can be run on the 100-word E101 can be run without change on a 220-word E101.

When utilizing more than 100 locations, the "H 3 b" and "S 3 b" instructions are used to start in the proper band and to go from one band to the next.

Obviously, the expanded memory E101 is a more powerful computing tool than the 100-word machine. But it is important to recognize that the superiority of the 220-word E101 stems from two quite different sources: *memory capacity* and *control.* While the extra bands on the drum permit the solution of problems requiring increased storage capacity, the extra band switch provides a third level of switch control. Experience has shown that the logical structure of many problems requires the extra control provided by the M switch of the 220-word machine, even though the storage requirements are well within the capacity of the 100-word E101. For example, when both the sine and cosine of an angle must be computed, a single loop can be used for both jobs, if the constants for the sine and cosine subroutines are stored in the corresponding memory locations of two of the bands.

EXPANDED MEMORY INSTRUCTIONS

PINBOARD AREA 1	2	3	
H	3	b	**Home Band Switch**—Home the band switch to position "b"; $0 \leqq b \leqq 3$. (This instruction may be used similar to "H 0 b" or "H 1 b" to home the M switch prior to stepping it, or it may be used merely to select one of the four 40-word bands. "Y" may be pinned in the 3rd area instead of a definite number, allowing the operator to select the band manually by means of the Y keys on the keyboard.)
S	3	b	**Step Band Switch**—Step the band switch once; if $M \neq b + 1$, execute the next instruction. If $M = b + 1$, skip the next instruction and execute the instruction after the next instruction ($0 \leqq b \leqq 3$).
H	4	—	**Accumulator Setting of E Switch**—Increase the E switch setting by the number in the least significant digit position of the accumulator. (Since "H 4" *increases* the setting of the E switch, the usual way of *homing* the E switch to the least significant digit of the accumulator is to first home it to zero by the instruction "H 0 0" and then increase it to the number in the accumulator by the instruction "H 4.")
H	5	—	**Accumulator Setting of F Switch**—Same using F switch.
H	6	—	**Accumulator Setting of E and F**—Same using E and F.
H	7	—	**Accumulator Setting of Band Switch**—Same using M switch.

Included as standard equipment with the 220-word memory E101 is the ability to home the band switch and the E and F switches to a number in the E101 accumulator. This ability to allow the program steps to make use of numbers in the memory is not available in the basic E101 except as an optional adjunct (see next section).

E101 220-WORD MEMORY MAP

Page No.________

Problem________________ Programmer________________

Band ______

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										

Band ______

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										

Band ______

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										

Band ______

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										

	0	1	2	3	4	5	6	7	8	9
4										
5										
6										
7										
8										
9										

ED-1276

ACCUMULATOR SETTING OF E AND F SWITCHES

As explained in the section on the expanded memory, the ability to set the E, F and band switches from the least significant digit of the accumulator is standard with the 220-word memory E101. The same feature (without the band switch) is available as an optional adjunct to the 100-word machine.

There are a number of places where this feature can be used to advantage, particularly in those problems where greater flexibility in the use of the E and F switches is desired. One important use is in *distribution,* both through the keyboard and via punched tape.

When distributing data through the keyboard to random memory addresses, the programmer has a choice of having the operator depress X and Y keys to designate the address, or having the operator index the address in the two right hand columns of the main keyboard along with the data. In the latter case, the address would appear in the accumulator all the way to the right with the units digit in the least significant place and the tens digit one place to the left. The F switch can be set to agree with the units digit of the address immediately. Then the entire number in the accumulator can be shifted one place to the right and the E switch set to the tens digit. One advantage of using this method of random address selection over the use of X and Y is that the address, which frequently corresponds to a code number, prints on the printed report along with the data. A cipher split can be set up in the carriage control panel so that there is space between the number and the address in the desired column.

When data coming in on punched tape is distributed to random memory addresses, the programmer has a choice of following each piece of data with a "W a b" instruction, or including the address as the right-hand portion of the number and thereby setting the E and F switches by means of this special feature. A complete program for random distribution from tape appears in Part III. This method is frequently used when the address corresponds to a code number and thus forms a natural part of the number itself. Tape preparation is also faster when the address is included with the number rather than in a separate "W a b" instruction.

PINBOARD AREA 1	2	3	Instructions for Accumulator Setting of E and F Switches
H	4	—	**Accumulator Setting of E Switch**—Increase the E switch setting by the number in the least significant digit of the accumulator. (Since "H 4" *increases* the setting of the E switch, the usual way of *homing* the E switch to the least significant digit of the accumulator is to first home it to zero by the instruction "H 0 0" and then increase it to the number in the accumulator by the instruction "H 4".)
H	5	—	**Accumulator Setting of F Switch**—Same using F switch.
H	6	—	**Accumulator Setting of E and F**—Same using E and F.

THE V SWITCH

The V switch is an optional device which can be used to relieve the operator of certain control responsibilities. It is helpful in those problems where there is a strict correspondence between the kind of numbers computed and the positions in which they are printed. The V switch is located on the keyboard-printer underneath the carriage, and its settings are controlled by projecting pins located in the carriage control panel. At any tab stop (columnar position), V can be set to any of its ten (0-9) values.

The V switch setting is used in the program by pinning V (located in pinboard area 3) in the desired instruction. As such, V is simply another 3rd area variable like E, F and Y, but whose value is a function of carriage position.

For example, if a problem requires keeping sums of keyboard entries, and the sum of the entries made in column 1 is to be stored in location 85, while the sum of the entries in column 2 is kept in location 89, then the control panel would be set so that V = 5 in column 1, and V = 9 in column 2. The instructions

```
K
+ 8V
W 8V
```

could then be used to make and accumulate keyboard entries.

The advantage of using V stems from the fact that to position the carriage in the proper column for each keyboard entry *requires a motor bar selection anyway.* If the previous carriage motion was due to a print instruction, P a _, then the motor bar was selected by the value of a. If the previous carriage motion was due to a keyboard entry, then the motor bar was selected by the operator. In either case, using the V switch eliminates the need of any *further* selection. The control required in the instruction comes from V; the value for V is determined by carriage position; and carriage position is determined by motor bar selection.

Without the V switch, XY key settings are usually employed to effect this kind of control.

PINBOARD AREA 1	2	3	USING THE V SWITCH IN INSTRUCTIONS
_	a	V	**Using the V setting**—Execute the instruction pinned in the first area of the pinboard using the carriage control panel setting of "V" in the third area. Any value for V ($0 \leqq V \leqq 9$) can be set up in the carriage control panel in any column position.

NOTE: Since V is set by mechanical pins located only at tab stop positions, V has no value in between tab stops. Therefore, an interlock circuit is used to prevent the E101 program from starting after a P or K instruction *until the carriage reaches the next column position.* Without the V switch, the E101 program continues while the carriage is traveling to its next position.

PART III PROGRAMMING AIDS

DECIMAL POINT SCALING

The machine decimal point ($_\wedge$) on the E101 is fixed at the extreme left of the 11-column keyboard. However, in the accumulator and memory, which have capacity for 12 digits, it is located at the left between the 1st and 2nd digits ($x_\wedge$xxx xxx xxx xx).

In dealing with problems on the E101, one must distinguish between the fixed point of the E101 ($_\wedge$) and the decimal point of the number itself (.). When a number is entered into the E101, it may or may not be entered with its decimal point (.) coinciding with the machine decimal point ($_\wedge$). An exponential type of notation is used to show the relationship of the actual point with respect to the machine point. For example, if the number 0.34 is entered in each of the following ways, it is said to be scaled as follows:

$0_\wedge$ 340 000 000 00—scaled $\times 10^{0}$
$0_\wedge$ 000 000 000.34—scaled $\times 10^{-9}$
$0_\wedge$ 000.340 000 00—scaled $\times 10^{-3}$
$.3_\wedge$ 400 000 000 00—scaled $\times 10^{+1}$

(Notice that in each case the exponent corresponds to the number of places between the two decimal points. A negative exponent indicates places to the right, and positive exponents places to the left, of the machine decimal point.)

A column is provided on the E101 programming sheet for keeping track of the scaling factor at each step of a problem.

Numbers to be added or subtracted must be scaled at the same power of ten. If they do not appear this way in the problem, they must be shifted left or right ("A 1 b" or "A 2 b") before being added or subtracted.

In multiplication, the exponent of the product is the sum of exponents of the two factors. Thus if two numbers each scaled $\times 10^{-3}$ are multiplied together, their product will be scaled $\times 10^{-6}$. If a uniform scaling is desired, the product can then be shifted 3 places to the left so that it will be scaled $\times 10^{-3}$ in alignment with the original factors.

In division, the exponent of the quotient is the exponent of the dividend less the exponent of the divisor. Thus if a number scaled $\times 10^{-3}$ is divided by a number also scaled $\times 10^{-3}$, the quotient will be scaled $\times 10^{0}$. Here again the answer can be shifted 3 places, this time to the right, so that it will be scaled $\times 10^{-3}$, the same as the dividend and divisor.

In some cases it is advisable to shift one of the terms *before* multiplying or dividing to make sure significant digits are not lost. For example, rather than divide a number scaled $\times 10^{-3}$ by another scaled $\times 10^{-3}$ and shift the answer 3 places to the right so that it also is scaled $\times 10^{-3}$, it might be better to shift the dividend 3 places to the right before dividing so that it is scaled $\times 10^{-6}$. The resulting quotient will then be scaled $\times 10^{-3}$ without further shifting.

Care must be taken at all times to prevent the loss of significant digits. In addition, subtraction, and division an overflow ALARM results if the answer $\geqq 10.0 \times 10^{0}$.

An overflow to the left cannot occur in multiplication, due to the location of the decimal point in the B Register. Significant digits may be unintentionally lost on the right, however, if care is not taken to have the numbers as far left in the B Register and memory as necessary.

Examples:

(12×10^{-6}) × (12×10^{-5})
$0_\wedge$ 000 012.000 00 × $0_\wedge$ 000 12.0 000 00
= (144×10^{-11})
= $0_\wedge$ 000 000 001 44 OK

(12×10^{-6}) ÷ (12×10^{-5})
$0_\wedge$ 000 012.000 00 ÷ $0_\wedge$ 000 12.0 000 00
= (1×10^{-1})
= 0. 1.00 000 000 00 OK

(12×10^{-6}) × $12 \times 10^{-6})$
$0_\wedge$ 000 012 000 00 × $0_\wedge$ 000 012.000 00
= (144×10^{-12})
= $0_\wedge$ 000 000 000 14(4) Digit lost (no ALARM)

(12×10^{-6}) ÷ (5×10^{-7})
$0_\wedge$ 000 012.000 00 ÷ $0_\wedge$ 000 000 5.00 00
= $(2.)4_\wedge$ 000 000 000 00
= $(2.4_\wedge \times 10^{+1})$ Digit lost (ALARM)

There are two basic methods of scaling problems on the E101. The first is to enter all data through the keyboard with the decimal point in a *fixed* keyboard position and shift the numbers internally as needed by means of the A 1 b and A 2 b shift instructions. This method is easy for the operator and provides uniformity of printing. It is also simple for an inexperienced programmer to use. A good starting point is to select a *fixed* keyboard position for the decimal point based on the range of the input data. For example, if the quantities range from xx.xxx to xxxx.xxx, it might be well to fix the decimal point arbitrarily 4 or 5 places from the left of the keyboard so that the quantities are scaled by a factor of 10^{-4} or 10^{-5}. Then as calculations take place, the results can be shifted left or right as necessary. The programmer sometimes finds it necessary to revise the position of the fixed point to avoid loss of significant digits in subsequent operations, but at least he has a good starting point from which to work.

The second basic method of scaling is to scale keyboard entries so that, for the most part, each number is entered with the decimal point where needed. This method of *varying* the position of the decimal point requires fewer program instructions than the *fixed* method of scaling.

The programmer will find that many problems follow typical patterns in which there are certain sequences of operations performed. Generally, a pattern of scaling can easily be worked out for such problems. As an example, consider the polynomial evaluation $C_6x^6 + C_5x^5 + C_4x^4 + C_3x^3 + C_2x^2 + C_1x + C_0$ which consists of a series of successive multiplications and additions. If the values of C and x are such that they can be scaled $\times 10^0$ without losing significant digits, there is no scaling involved at all.

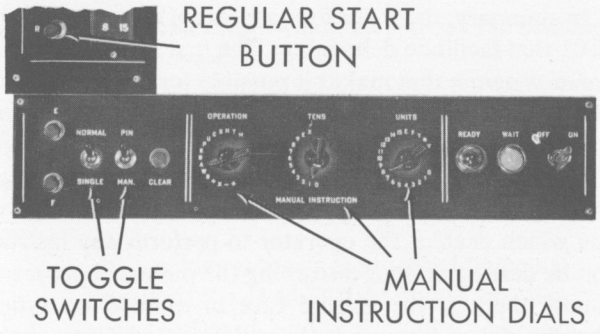
If scaling is necessary, either of the two basic methods can be used. If the *fixed* keyboard method of scaling is used, all values of C and x are entered times a power of ten other than zero with the decimal point in a *fixed* keyboard position. The resulting products are therefore scaled times different powers of ten and must be shifted before being added. The programmer has a choice of shifting each term immediately after multiplying or waiting until it is time to add them together. For example, if x and all values of C are entered scaled $\times 10^{-1}$, then C_6x^6 will be scaled $\times 10^{-7}$, $C_5x^5 \times 10^{-6}$, $C_4x^4 \times 10^{-5}$, etc. Before adding them together, they must be shifted right or left so that all of them are scaled at the same power of ten.

The other method of scaling this problem is to enter x scaled at a power of ten other than zero and enter the constants scaled at *varying* powers of ten so that when the constants are multiplied by the powers of x, the resulting products are all scaled by the same power of ten and can be added together without shifting. For example, if x is scaled $\times 10^{-1}$, then x^2 will be scaled $\times 10^{-2}$, $x^3 \times 10^{-3}$, $x^4 \times 10^{-4}$, $x^5 \times 10^{-5}$, and $x^6 \times 10^{-6}$. If the constants are entered so that C_0 is scaled $\times 10^{-6}$, $C_1 \times 10^{-5}$, $C_2 \times 10^{-4}$, $C_3 \times 10^{-3}$, $C_4 \times 10^{-2}$, $C_5 \times 10^{-1}$ and $C_6 \times 10^0$, the resulting products will all be scaled $\times 10^{-6}$.

The programmer will quickly become familiar with recurring patterns such as $\frac{ab}{c}$. Frequently scaling instructions are not necessary due to the fact that while multiplication shifts the answer to the right, division can be arranged to compensate by shifting it back to the left. There are other patterns similar to this where scaling is not necessary. Care must be taken at all times, however, to make sure significant digits are not lost.

DEBUGGING

A program can be checked out or debugged on the E101 in a relatively short time. The process is simplified by the externally stored program, and the easy-to-operate control panel. Two features of the control panel that are



used in debugging are *Single Step* operation and *Manual* control, both of which are described below.

Single Step Operation—Normally the E101 goes automatically from one pinboard instruction to the next without stopping. In checking out a program, it is often helpful to execute a single instruction at a time, stopping after each step before going on to the next. We call this Single Step operation as opposed to Normal operation. When an operator wishes to use Single Step operation, he moves the NORMAL/SINGLE toggle switch from the NORMAL to the SINGLE position. Then he touches the REGULAR START button which executes a single pinboard instruction. To execute the next instruction, he must again touch the REGULAR START button. The process is continued until the operator returns the toggle switch to the NORMAL position.

Manual Control—Frequently when checking out a program—particularly when using Single Step operation—it is deirable to perform certain operations that have not been pinned up in the pinboards. This is especially true where the operator wishes to print out and examine the contents of the accumulator. The operator can perform any instruction he desires at any point in the problem *without affecting the pinboard program* by first stopping automatic operation (i.e., calling for SINGLE operation), and then throwing the E101 out of pinboard control and into manual control.

To do this, he moves the PINboard/MANual toggle switch from the PINboard to the MANual position. Then he sets the three Manual Instruction Dials to correspond to the instruction he wants to perform. The three dials represent the three areas of the pinboard, and *any* instruction that can be pinned in a pinboard can be set up on the dials. In the case of a print instruction dial 1 should be set at "P" for "Print," and dial 2 at 1, 2, 3 or 4 depending on which motor bar the programmer wants activated. In an instruction which consists of only one or two characters (e.g., A 3 _ or B _ _), the manual switch(es) corresponding to the unused character(s) can be ignored. In the case of a manual Print instruction, the units area switch setting is irrelevant except that a P a O will, of course, effect a non-print. After the instruction is set up on the dial, the operator executes it by touching the REGULAR START button. To continue with the pinboard program, the operator returns the control toggle switch to the PINboard position.

In summary, there are four aspects of control over the E101 that facilitate debugging: The first is the *externally stored program* that makes it possible for the operator to alter the program merely by rearranging one or more pins in the removable pinboards. The second is *Single Step operation* which allows the operator to run through the program a step at a time. Next there is *Manual operation* which enables the operator to perform any instruction he desires without disturbing the pinboard program. Finally there is the special case of manual operation whereby the operator can *print* out the contents of the accumulator without affecting the pinboard program.

A step by step procedure for checking out a program follows. The first step should generally take place *not* at the E101 but at the programmer's desk. It consists of "running through" the program *on paper* using sample data and making a careful record of the intermediate and final results. This procedure is important, as it can save considerable time and effort later on. Experienced computer programmers agree that it is foolish to omit this step in hopes of "short-cutting" the checking-out procedure, since the steps that follow are much more difficult and time-consuming without the results of the "paper" run.

With the pinned-up program and the sample calculations, the programmer is ready to try out the problem on the E101. Using the sample data, he runs through the problem on the E101 and compares the answers with the pre-calculated results. If, as is frequently the case, the results do not agree, the programmer must debug, that is find the source of error in his program.

Basically, debugging involves going through the program a step at a time and printing out the contents of the accumulator after each step without disturbing the pinboard program:

1. Set the Manual Instruction dials to any of the four print operations: P1, P2, P3 or P4.
2. Set the NORMAL/SINGLE toggle switch to the SINGLE position. (Leave the second toggle switch in the PINboard position.)
3. Start the machine at the beginning of the problem by depressing the appropriate START button (usually No. 1).
4. Using the same data as before, perform the first pinboard instruction by depressing the REGULAR START button (marked "R"). The neon lights indicate which program step has been executed. The entire process can be followed by reading the instructions from a program sheet or directly from the pin settings in the pinboards.
5. Print out the result of the operation by:
   a) Setting the PINboard/MANual toggle switch to the MAN position and
   b) Pressing the REGULAR START button.

   This takes the E101 out of pinboard control and performs whatever instruction has been set up on the Manual Instruction dials—in this case, a print operation.
6. Return the E101 to pinboard control by setting the PINboard/MANual toggle switch back to the PIN position.
7. Perform the next pinboard instruction and print out the new result by repeating steps 4, 5 and 6 above. Check each result with the pre-calculated results. Continue until the error or omission has been located and corrected.

With a little experience, the programmer will find that it is not necessary to print out the contents of the accumulator after *every* instruction. K and P instructions print automatically anyway, and W, B, U, C, S, and H instructions leave the accumulator unchanged. Some programmers print out results only at selected points in the program until they narrow down the portion of the program where the error is located. Then they print out the contents of the accumulator after *every* instruction which affects the accumulator until the error is tracked down. When the error is located during the course of the "single step" operation, it can frequently be corrected in a minute or two by rearranging one or more pins in the pinboards.

As the program is checked out, the programmer can tell which program step is being executed at any given time by observing the neon pinboard and step lights on the control panel. The first column of lights, numbered from 1 to 8, indicates the pinboard number while the second column, numbered from 0 to 15, indicates the step number.

These same lights can be made to indicate the positions of the E and F switches, allowing the programmer to check his H and S instructions. To determine the E switch position at any given time, the operator depresses the button on the left of the control panel marked "E." While he holds down the E button, the "step" lights (0 to 15) will indicate the E switch position. To determine the F switch position, the operator depresses the button marked "F." To determine the band switch setting (on a 220-word memory E101), he depresses both buttons simultaneously. As soon as the operator releases the button (or buttons), the lights again indicate the step in the pinboard program.

TIMING

Sometimes programmers want a time appraisal of a program. The reason might be to evaluate several approaches to a problem, or to determine whether the problem is an "economical" one for a computer. The following time parameters on various operations, although not given in complete detail, are sufficient for making good appraisals:

Addition type instructions (+, −, R, W, B, A) —1/20 sec.
Average multiplication or division instruction —¼ sec.
Average transfer instruction—½ sec. (varying with how large a "jump" is involved).
Stepping instruction—1/20 sec.
Average homing instruction—0.3 sec.
Print or Keyboard instruction—0.6 sec.
Iinstruction on tape—3/20 sec. in addition to time for same instruction on pinboard.
Data entry from tape—20 characters per sec.
Tape output—20 characters per sec.

Unlike larger computing systems, every instruction on the E101 takes at least two drum revolutions. The techniques of minimum access programming cannot be applied to a machine like the E101.

The only place—but a significant one—where time can be saved by arrangement of the program is the transfer operation. This is because transfers are carried out by stepping switches. For example, each pinboard is controlled by its own 16-position stepping switch (these switches are physically the same as the E and F switches, but they are not accessible via instructions in the program). If a pinboard switch is at step 11 and a transfer operation calls for its going back to step 0, the switch must step through positions 12, 13, 14 and 15 before reaching step 0 to execute it.

Which pinboard is in control is also set by a switch, and the same considerations hold: a transfer to the *succeeding* pinboard takes about ¼ the time required to transfer to the *previous* pinboard

A discussion of the advisability of iterating within a pinboard, and an example of saving time by eliminating transfers appear in Part IV.

CHECKING CIRCUITRY

1. General

Like other general purpose digital computers, the E101 is provided with circuitry whose sole function is to check the operation of the components which do the computing. When this checking circuitry detects a machine malfunction, it stops the computation, and signals the operator accordingly.

Although erroneous results are sometimes caused by a component failure, they can also occur due to improper operation or programming of a computer: the operator may make an entry in the wrong columns of the keyboard, a pin in the pinboards may be misplaced, the program steps may generate an overflow, etc. Computers usually have circuitry which detects certain of these human failures, also. The overflow alarm on the E101 is an example of this kind of checking circuit.

These sections are intended to help the E101 user in quickly diagnosing and correcting the various conditions —internal and external—which may cause the checking circuits to stop computation.

2. Trouble Symptoms—ALARM Light

The basic trouble signal on the E101 is the ALARM, whose occurrence is indicated by the following conditions:

a) The machine stops on the instruction being executed; the HALT light comes on; the neon pinboard and step lights indicate where in the program the ALARM occurred.
b) The ALARM light comes on; touching the CLEAR button will turn off the ALARM light, permitting the START buttons to function; if the operator doesn't touch the CLEAR button within the first few seconds, a repeating gong sounds.

The condition which caused the ALARM is indicated by other factors, which will be covered in detail later in this section.

3. Trouble Symptoms—Machine Standstill

Another symptom of trouble is a computer stop or standstill, in which the E101 stops, although neither the ALARM light nor the HALT light comes on. To get the computer going again the operator usually touches the CLEAR button (which turns on the HALT light), then corrects the condition and continues by pressing one of the START buttons. The various causes of a standstill are discussed in detail below.

PROGRAMMING AND OPERATING ERRORS

1. The Overflow Alarm

Overflow in the E101 is signalled by the ALARM condition occurring on one of the following instructions:

a) If the ALARM occurs on a +, −, or ÷ instruction, the value of the sum, remainder, or quotient, respectively, is $\geqq 10$ (10×10^0).
b) If the ALARM occurs on a B instruction, it means that the contents of the accumulator is $\geqq 1$ (1×10^0).

In either case, the overflow is due to improper scaling.

If the program has not been checked out, it must be corrected to compensate for the error.

If the problem has been debugged, examine the entire pinboard program thoroughly for a pinning error.

If no pinning errors are found, check the scaling of the numbers entered through the keyboard.

If keyboard entries were made properly, print out the contents of the memory (or those locations used in the current problem) and check your stored constants and initial input values.

If none of these steps reveals the trouble, repeat the program leading up to the ALARM; if the ALARM recurs, go into SINGLE operation and recheck those instructions which generate the number causing the overflow. If you cannot isolate the instruction(s) this way, and the ALARM continues to occur, contact your ELECTRODATA Field Engineer.

2. Machine Standstill

A machine standstill is the result of an attempt to operate the E101 without first putting it into proper operating condition. There are two main causes of a standstill:

a) *Computer not prepared for operation*—the following check list will help correct any such oversights:
   1) Make sure that the READY light is on. If not, check that *both* the E101 key switch and your circuit breaker are on. If both switches are on, but there is no READY signal, call your Field Engineer.
   2) Make sure that the PINboard/MANual toggle switch is in the PINboard position.
   3) See that the NORMAL/SINGLE switch is in the NORMAL position.
   4) Check that the right carriage control panel is in the keyboard-printer; check the Schedule knob setting; make sure that the panel is seated correctly in the carriage; operate the TAB and RETurn keys on the keyboard-printer to insure that the carriage is stopped in one of its tabular column printing positions.
   5) Make sure that the PINBOARD DISCONNECT key in the extreme lower right-hand corner of the keyboard-printer in its up, or released, position.
   6) Check the position of the PROGRAM lever in the upper right-hand corner of the keyboard-printer; it should be forward, in its PROGRAM position.
   7) Operate the carriage open-close key to make sure the front insert is closed.
   8) Check keyboard to see that keys are set in their fully-depressed position; use Error key to release stuck keys.

b) *Attempt to execute an instruction with missing pin(s).*
   1) If the missing pin is in the operation area (#1), just insert the pin. The E101 will execute the instruction and continue with the program.
   2) If area 2 and/or 3 has a pin missing, touch the CLEAR button, insert the missing pin(s), execute a MANual transfer back to the instruction in question, return to PINboard control, NORMAL operation, and touch the REGULAR START button.
   3) If the E101 is sent to an instruction in a pinboard, and the entire pinboard is missing, simply insert the pinboard.
   4) Note that in memory addresses, a pin in area 3 placed between b = 10 and b = 15 will act like a missing pin.

3. Double Pinning

The E101 does not have circuits to check for the insertion of *extra* pins in an instruction (e.g., B 7 15, A 3 9). It simply ignores such extra pins, and executes the instruction as if the redundant pins weren't there (B _ _, A 3 _). This feature is sometimes quite useful; to give the operator the choice, using the X key, of whether or not to carry out a left shift of 6 places, we could use an A X 6 instruction. When the operator sets X = 1, the shift is executed (A 1 6); when the operator sets X = 3, an absolute value instruction is executed instead (A 3 _).

But when more than one pin appears *in the same area* of an instruction, trouble develops. The effect of a double pinning is to short the two lines pinned, *so that every instruction in the pinboards calling for either of the shorted lines is shorted, too.* For example, if the program calls for an R 7 9 instruction in step 11 of pinboard 8, and the operator mistakenly double-pins R and U (giving RU 7 9), then there is as good a chance of executing a U 7 9 as there is an R 7 9.

> Furthermore, *any* R instruction appearing *anywhere* in the program *is liable to be interpreted as a U,* and *any U instruction as an R.* Instructions selected by the manual switches are subject to the same conditions: a double pin will disturb any instruction which calls for one of the shorted lines.

Thus, although there is no circuitry to detect double pinning directly, the presence of a double pin manifests itself very quickly. Symptoms are overflow alarms, machine standstill, skipping or repeating an instruction or a whole sequence of instructions, inability to leave a pinboard, or to get out of a loop, inconsistent printing patterns, impossible printed results, failure to load or read out properly, etc.

When faced with such symptoms, a visual check of the pinboards will ususally reveal the double pins. If double pinning cannot be found visually, remove all the pinboards, and execute the instruction manually. If the manual instruction won't work properly, call the Field Engineer. If the manual instruction works with the pin-

boards removed, replace the pinboards one at a time, *testing the manual instruction after each replacement.* If trouble recurs, the error is in the last pinboard replaced.

MACHINE MALFUNCTION

1. Print ALARM

The E101 has a print check circuit which translates the type bar positions into pulses just after printing, and echoes these pulses back against the number in the accumulator. If comparison does not occur, the machine lights the ALARM signal. This is distinguished from an overflow ALARM by the fact that it occurs on a print instruction.

When a print ALARM occurs it can be assumed (in the absence of other ALARMS) that the number in the accumulator is correct. At worst, the printed results will contain the incorrect number.

When the printing mechanism fails, it is very often a temporary condition, which can be remedied by repeated exercise of the mechanism. Executing several manual Print instructions often clears up the mechanical difficulty, and printing will continue normally thereafter.

Whether or not the difficulty proves to be temporary, the Field Engineer should *always* be notified of the occurrence of a machine malfunction condition.

2. Pulse Sequence Check

Information is stored on the drum of the E101 in pulse-coded decimal form. Each digit consists of 9 bits. All 9 magnetic spots oriented one way is a 0, all nine the opposite way is a nine (if the digit is a 5, 5 magnets point one way, the other 4 point in the opposite direction, etc.). When the digit is not zero, definite rules apply as to the position of the non-zero magnets: they must appear in a dense sequence, and they must be adjacent to the end-digit control pulse. Any other configuration is forbidden,

and indicates a machine malfunction. A condition like a sudden drastic change in line voltage can cause a pulse to be lost or picked up. Such an occurrence is checked by having the initial pulse of each pulse sequence trigger an independent pulse generator, whose output is gated against the output of the pulse amplifier tube each clock time, until the end digit control pulse resets the generator. Failure of any of these outputs to compare sets the ALARM circuit, and lights one or both of the amber lights behind the ALARM signal.

When a Pulse Sequence ALARM occurs, the operator should immediately notify the Field Engineer. Have available as much information as possible regarding the state of affairs when the alarm occurred: the kind and location of the instruction being executed, the memory locations involved, the size of the numbers being manipulated, etc. This information may enable him to advise you of temporary steps you can take to continue your computation while the Field Engineer is on his way to your installation.

CHECKING INPUT DATA

In using the E101 with keyboard input of data, it is very important that there be a reliable method of detecting and correcting input errors. Some of the methods being used successfully by various E101 users are as follows:

Visual Check

After entering the data, the operator checks visually to make sure no errors have been made. A variation of this method which one E101 user has found effective is to have the data pre-entered by hand or typewriter on the report with space directly below each number for the E101 to print. After entering the data through the E101 keyboard, it is a very simple matter to compare the number printed by the E101 with the pre-printed and hand-written number directly above it.

Proof Totals

A popular way of checking the accuracy of keyboard input, particularly in accounting applications, is to have the E101 total the data and compare the result to a predetermined proof total. This operation can be programmed in the ususal way, or if desired, the data can be accumulated in the E101 crossfooters and registers.

Correction of Errors

The correction of an input error can be done in a number of different ways, depending mainly on *when* the error is detected. If the operator realizes his mistake *before* he touches the motor bar, he can remedy the situation immediately by depressing the Error key (marked "E") at the right of motor bar 2 and re-entering the number correctly.

If he discovers the error *after* the motor bar has been touched, the thing he must determine is whether the number has merely been stored in the memory or has actually

been used in calculations. Frequently the error is detected after the number has been stored but *before* any calculations have taken place. Having stopped the machine by throwing the switch to SINGLE operate, the operator can easily correct the situation by putting the E101 into MANUAL control, entering the correct number, and writing it into the appropriate memory location.

If the error is detected *after* a series of calculations, the results of the calculations can often be corrected by means of a "correction program" consisting of pinboard instructions which replace the incorrect results with correct ones. Generally, the "correction program" runs through the same calculations as before using the incorrect quantity again but with the sign reversed. This has the effect of compensating for the results produced by the wrong numbers. Next the *correct* data is entered into the keyboard and the calculations repeated.

PROGRAMS AVAILABLE TO E101 USERS

The programs for basic functions (such as square roots, logarithms and trigonometric functions) which are used freqeuntly in scientific, engineering and statistical computations are generally called "subroutines" for electronic digital computers. One can, to a considerable extent, program these computations once and for all and have them available for use as part of the program in any problem in which basic functions are required.

For the convenience of our customers we have programmed the most common of these basic functions and have compiled the programs into a booklet entitled *Handbook of Subroutines and Subroutine Methods for the* ELECTRODATA *101,* which is available to all E101 users. A list of the programs included in the subroutine booklet appears below.

In addition, the ELECTRODATA *Division Applied Mathematics Department will advise E101 users regarding their larger problems in mathematics, and has available programs for the "standard" numerical analysis problems, such as matrix inversion, roots of polynomials, linear regression, and others.*

Subroutines Available to E101 Users

$\sin u$ $(-\frac{\pi}{2} \leqq u \leqq \frac{\pi}{2}$, u in radians)

$\sin u$ $(-90° \leqq u \leqq 90°$ or $-\frac{\pi}{2} \leqq u \leqq \frac{\pi}{2}$, u in radians or degrees)

$\sin A$ (for large angles, A, in radians or degrees)

$\cos u$ $(-\frac{\pi}{2} \leqq u \leqq \frac{\pi}{2}$, u in radians)

$\cos u$ $(-90° \leqq u \leqq 90°$ or $-\frac{\pi}{2} \leqq u \leqq \frac{\pi}{2}$, u in radians or degrees)

$\cos A$ (for large angles, A, in radians or degrees)

$\tan u$ $(-\frac{\pi}{4} \leqq u \leqq \frac{\pi}{4}$ or $-1 < u < 1$, u in radians)

$\arccos u$ and $\arcsin u$ $(-\frac{1}{2} \leqq u \leqq \frac{1}{2}$ or $-\frac{\sqrt{2}}{2} \leqq u \leqq \frac{\sqrt{2}}{2}$, result in radians)

$\arccos u$ and $\arcsin u$ $(0 < u < 1$, result in radians)

$\arctan u$ $(-1 < u < 1$, result in radians)

$\arctan u$ $(0 < u < 999$, result in radians)

e^u $(-1 \leqq u \leqq 1)$

e^{-u} $(0 \leqq u < 10)$

10^u $(0 \leqq u \leqq 1)$

$\log_{10} u$ $(1 \leqq u \leqq 10)$

$\log_e u$ $(1 \leqq u \leqq 10)$

$\sinh u$ $(-4.5 \leqq u \leqq 4.5)$

$\cosh u$ $(-4.5 \leqq u \leqq 4.5)$

$\tanh u$ $(-2 \leqq u \leqq 2)$

$\sqrt{u}$

Multiplication of complex numbers

$(a + bi)\ (c + di)$ for $-0.9999 \leqq a, b, c, d \leqq 0.9999$

Division of complex numbers

$\frac{(a + bi)}{(c + di)}$ for $-0.9999 \leqq a, b, c, d \leqq 0.9999$

STANDARD OPERATING ROUTINES

The three most common operating routines are clearing, loading and printing the contents of the memory. Because these routines are used so frequently, their programs are given below. In each case the routine is programmed for the maximum number of memory locations (100 or 220); the same programs can be used for fewer operations. The approach in all three cases is based on programming the given problem *in as few steps* as possible. While this is the most popular approach, other methods (some of which are described in Part IV) may be used.

The similarity among the three routines is immediately apparent. Because of this similarity, it is possible to program a multipurpose routine that can be used interchangeably for all three opeartions merely by moving a few pins. This routine is also shown below.

1) Clearing Entire Memory

0	R 9 9	Read contents of memory location 99 into accumulator
1	— 9 9	Subtract contents of memory location 99 from accumulator
2	H 0 0	Home E switch to zero
3	H 1 0	Home F switch to zero
4	W E F	Write contents of accumulator (zero) into memory location E F
5	S 1 9	Step F switch once each time through routine until F passes 9; then transfer to step 7.
6	U 0 4	Transfer back to step 4 and repeat routine with new value for F.
7	S 0 9	Step E switch once each time E101 reaches this step (once every 10 times through routine); when E = 10, transfer to step 9.

(steps 0 and 1: to generate zero)

8	U	0 3	Transfer back to step 3 and repeat routine with new value for E.
9	-	- -	

2) Loading Entire Memory

0	H	0 0	Home E switch to zero.
1	H	1 0	Home F switch to zero.
2	K		Transfer contents of Keyboard into accumulator when operator touches motor bar.
3	W	E F	Write contents of accumulator into memory location E F.
4	S	1 9	Step F switch once each time through routine until F = 10; then transfer to step 6.
5	U	0 2	Transfer back to step 2 and repeat routine with new value for F.
6	S	0 9	Step E switch once each time E101 reaches this step (once every 10 times through routine); when E = 10, transfer to step 8.
7	U	0 1	Transfer back to step 1 and repeat routine with new value for E.
8	-	- -	

Note: When the data to be loaded is on tape, step 2 is T _ 12 instead of K.

3) Printing Contents of Entire Memory

0	H	0 0	Home E switch to zero.
1	H	1 0	Home F switch to zero.
2	R	E F	Read contents of memory location E F into accumulator.
3	P	2	
4	S	1 9	
5	U	0 2	Same as steps 4 to 7 in loading routine
6	S	0 9	
7	U	0 1	
8	-	- -	

4) Multipurpose Routine for Clearing, Loading and Printing

By changing steps 4 and 5 for each operation, the following routine can be used for clearing, loading and printing the entire memory (100 words). Because all three operations are used frequently, some programmers find it convenient to keep a multipurpose pinboard pinned up at all times. It is helpful to have a template stamped in one color with the exception of steps 4 and 5 which could be stamped in three different colors, representing the three operations.

0	R	9 9	
1	—	9 9	
2	H	0 0	
3	H	1 0	
4			
5			
6	S	1 9	
7	U	0 4	
8	S	0 9	
9	U	0 3	
10	-	- -	

CLEAR	LOAD	PRINT
W E F	K	R E F
W E F	W E F	P 2

5) Loading the 220-Word Memory

0	H	3 0
1	H	0 0
2	H	1 0
3	K	
4	W	E F
5	S	1 9
6	U	0 3
7	S	0 3
8	U	0 2
9	S	3 3
10	U	0 1
11	-	- -

The above routine will load the 160 locations of the 4 switchable bands of the 220-word memory (00-39 in bands 0, 1, 2 and 3). The remaining 60 words of the heartband (40-99) can also be loaded with the same routine by pinning a U 0 2 in step 11 (after entering the 220th number, the operator would touch the CLEAR button and then a START button going to the rest of the program).

6) Loading from Punched Tape Input

When a data tape has been prepared to use with the E101, it can be read and loaded using any of the other loading routines by simply substituting a T _ 12 instruction for the keyboard instruction given.

When data in random sequence on tape is to be distributed in the E101 memory, the accumulator setting of E and F switches can be used effectively. Assume that a 6-digit amount (xxxxxx), followed by its 2-digit classification or identification number (ab), appears in each word on tape:

[xxxxxxab]

The program and its operation would be as follows:

	PROGRAM			EXPLANATION	CONTENTS OF ACCUMULATOR	SWITCH SETTINGS E	F
0	T	—	12	read tape word into accumulator	0000xxxxxxab	?	?
1	H	1	0	home F switch to 0	0000xxxxxxab	?	0
2	H	5	—	add b to F	0000xxxxxxab	?	b
3	A	2	1	shift accumulator right one place	00000xxxxxxa	?	b
4	H	0	0	home E switch to 0	00000xxxxxxa	0	b
5	H	4	—	add a to E	00000xxxxxxa	a	b
6	A	2	1	shift accumulator right one place	000000xxxxxx	a	b
7	+	E	F	add previous sum to xxxxxx	new sum	a	b
8	W	E	F	store new sum in ab	new sum	a	b
9	U	0	0	repeat with next tape word	new sum	a	b

PART IV PROGRAMMING STRATEGY

GENERAL

Programming a problem for an electronic computer consists of two basic parts: first, determining the proper approach or strategy and second, coding. With the ELECTRODATA 101, the coding part is quite straightforward because the E101 program language is so close to the language of arithmetic itself. Determining what approach or strategy to use requires somewhat more skill and imagination on the part of the programmer, and is an interesting and challenging experience. The main factors to consider are the nature of the problem and the capacity of the computer. Each problem and each computer has its own special features.

The job of the programmer is to fit the problem into the computer. Where some aspect of the problem exceeds the capacity of some feature of the machine, the programmer must try to compensate for this by the use of the machine's other features (e.g., if his first approach takes too long for a solution to be reached, he may use more program steps to cut down running time).

In larger computer systems, the main machine limitations are storage capacity (data and program), and operating speeds. In an externally programmed computer like the E101, the storage capacities for data and programs are not interchangeable, and must be considered separately: the E101 programmer must fit the stored data into the drum, the instructions into the pinboards, within the problem solution time requirements. He can use the 220-word memory and/or the tape reader to increase data storage, he can change pinboards, or use instruction tapes to increase program capacity, etc.

How long is the problem? How often is it going to be run? These questions are important since they give the programmer an idea of how much time and effort to spend on the problem. If it is a relatively short problem that is going to be run only once or twice, there is not much point in developing a highly sophisticated approach that will save running time and perhaps reduce the number of program steps. On the other hand, it makes sense to concentrate on programming strategy that will save running time if the problem is a fairly long one that will be run many times.

Point of emphasis varies from problem to problem and from program to program. Oftentimes, the most important consideration is the reduction of the number of programming steps. Sometimes it is "increasing" the size of the memory. In other cases, it is cutting down the running time, while in still others, it is ease of operation. In many problems it is a combination of two or more of these factors. The most important part of the programmer's job is placing the emphasis where it belongs. At all times a reasonable balance between programming time and running time should be maintained.

Part IV is devoted to programming ideas or techniques that E101 programmers have found helpful. Some save on programming space, some running time, and others on memory space. Many of them are merely means of using certain instructions (or combinations of instructions) in a way that is not too obvious. Some of the ideas presented here were "developed" by our staff of Sales Representatives, Sales Technical Representatives, and Mathematical Analysts, while others were contributed by our customers. Although we are thus depriving you of the fun of discovering them for yourself, we hope they will save you time, and make your programming job a more interesting and profitable one.

The various ideas and techniques discussed here tend to fall into fairly distinct categories:

A. *Input-Output Ideas*—The ideas in this section are commonly used in input-output routines.

B. *Logical Subroutines*—The programming ideas included here are called "logical subroutines" since they accomplish some frequently needed logical manipulations. For example, although the E101 program language does not include an equality test as a single instruction, one can be programmed with two instructions, as described in this section. This category also includes such topics as split register storage and counting.

C. *Special Information on Basic Instructions*—This section contains ideas on how some of the basic programming instructions can be used in important ways. Because of the nature of the ideas, they are better introduced here in a separate section rather than as part of the concise descriptions given in Parts I and II.

D. *Algebraic Manipulations*—Some aspects of ordinary manipulation of algebraic expressions are such that a particular approach is better for computer programming than any of the other possible approaches. Some examples of this idea are given in this section.

A. INPUT-OUTPUT IDEAS

1. Loading a number of amounts into the memory where the total number is not divisible by ten

Using the E or F switch, it takes 5 program steps to load up to 10 amounts into the memory. Using both E and F, it takes 8 steps to load 20, 30, 40, etc., up to 100 amounts into the memory. A normal way of loading a number of amounts where the number is not divisible by 10, for example 37, is to break the problem into two parts: loading the first 30 amounts, and then the last 7. This would involve 8 steps for the first part, plus 5 for the second, making

a total of 13 program steps. The same problem can be programmed in just 9 steps by loading the 7 amounts *first* and then the 30 amounts, leaving the blank memory locations *at the beginning instead of at the end.* The two routines are shown below for comparison.

	1 37 Amounts in Memory Locations 00 to 36	2 37 Amounts in Memory Locations 03 to 39
0	H 0 0	H 0 0
1	H 1 0	H 1 3
2	K	K
3	W E F	W E F
4	S 1 9	S 1 9
5	U 0 2	U 0 2
6	S 0 2	H 1 0
7	U 0 1	S 0 3
8	H 1 0	U 0 2
9	K	
10	W 3 F	
11	S 1 6	
12	U 0 9	

Another way of loading a number of amounts where the number is not divisible by ten is to store the numbers in a rectangular array. This approach works whenever the number of amounts being stored is a number that is not a prime (divisible only by itself) and can be factored into two parts *both* less than 10. If there are 59 amounts, for instance, this approach cannot be used since 59 is a prime number. If there are 68 amounts, the approach still cannot be used, for even though 68 is divisible by 4, the other factor, 17 ($4 \times 17 = 68$), is larger than 10. An example of a situation where the approach can be used is in loading 45 amounts ($5 \times 9 = 45$.) By loading the amounts in 5 rows of 9 each or 9 rows of 5 each, the routine can be programmed in just 8 steps.

Although the discussion here has dealt with memory loading, the same basic procedures can be used in clearing, printing, and performing arithmetic operations: any approach which will load numbers into the memory for you, will allow you to compute with those numbers, read them out, etc.

2. Entering an indefinite number of amounts through the keyboard

Some problems call for an indefinite number of amounts to be entered through the keyboard, each one used immediately in computation rather than being stored. An example would be the evaluation of the polynomial $x^3 - 2x^2 + 14x$ where x might be assigned any number of values. The usual way of programming such a problem is to form an iterative loop starting with "K," followed by the steps involved in the computation, and ending with a "U" that sends the E101 back to the "K" instruction. This type of loop works very nicely until the operator reaches the point where there are no more keyboard entries. Since the "U" instruction at the end of the routine *always* sends the E101 back to the "K" instruction, the operator will find the E101 waiting for a keyboard entry, but will have no more data to enter. The way to get out of the loop and into the next part of the program is to touch the CLEAR button. This turns off the KEYBOARD light and lights the HALT signal. With the E101 in the HALT condition, the operator can touch the proper START button which will take the E101 to the beginning of the pinboard where the rest of the problem is programmed.

3. Replacing the contents of each memory location with the contents of the succeeding memory location

Some problems require replacing the contents of each memory location with the contents of the succeeding memory location. Using row 5 as an example, the program for this operation is as follows:

0	H 1 0	Home F to zero
1	H 0 1	Home E to one
2	R 5 E	Read contents of 5E.
3	W 5 F	Write into preceding memory location.
4	S 2 9	Step E and F together until E passes 9.
5	U 0 2	Transfer back to read next memory location.

B. LOGICAL SUBROUTINES

The ideas included in this section are as follows:

1. Counting using the "C" instruction
2. Counting using the "S" instructions
3. Split register storage
4. Split register storage used in conjunction with X and Y keys for random access
5. Equality test

1. Counting using the "C" instruction

The "C" instruction is frequently used to keep count of the number of operations performed. To illustrate, assume that the instructions in pinboard 1 are to be performed 75 times. The numbers 0, 1, and 75 are stored in three memory locations (assume in this case locations 97, 98 and 99, respectively). Each time the counting routine is repeated, the "1" in memory location 98 is added to the number in 97 and the total written back into 97. Then the 75 stored in memory location 99 is subtracted from the total. The next instruction is a "C." If the answer is negative, indicating the routine has not been performed 75 times as yet, the C 1 0 sends the E101 back to pinboard 1 to repeat the routine. If

the answer is positive (or zero) indicating that the routine *has* been performed the required number of times, the "C" instruction goes on to the next step in the problem, where operation is continued.

The program steps required for the "count and compare" operation are as follows:

Pinboard	Step	
1	0	_ _ _
	•	
	•	
	15	U 2 0
2	0	R 9 7
	1	+ 9 8
	2	W 9 7
	3	– 9 9
	4	C 1 0
	5	_ _ _

Another way of programming this problem, starting with 74 in 97 and 1 in 98, is as follows:

Pinboard	Step	
1	0	_ _ _
	•	
	•	
	15	U 2 0
2	0	R 9 7
	1	– 9 8
	2	W 9 7
	3	C 2 5
	4	U 1 0
	5	_ _ _

In the above case the answer is always positive until the routine has been performed 75 times. Each time the E101 reaches the "C" instruction, it automatically goes on to the next instruction in the program, U 1 0, which transfers back to repeat the routine. After the 75th performance, when the answer becomes negative, the E101 transfers to step 5 in pinboard 2. Other variations include starting with a negative number in 97 and adding "1" to it until the total, which starts negative, passes through zero and becomes positive.

2. Counting using the "S" instruction

Many programmers use the "S 0 b" and "S 1 b" instructions exclusively for address modification. While this is their main function, they can also be used for counting. Since both E and F switches are 16-position counters, either switch can be used to count up to 16, or together they can be used to count up to 256.

In the following examples, a given routine is to be performed the number of times indicated:

65 Times	110 Times	96 Times	256 Times
H 0 1	H 0 1	H 0 1	H 0 0
H 1 1	H 1 1	H 1 1	H 1 0
Routine to be repeated	Routine to be repeated	Routine to be repeated	Routine to be repeated
S 1 13	S 1 11	S 1 12	S 1 15
U 0 2	U 0 2	U 0 2	U 0 2
S 0 5	S 0 10	S 0 8	S 0 15
U 0 1	U 0 1	U 0 1	U 0 1

Notice in the first three examples, where we start with "H 0 1" and "H 1 1," that the total number of times the routine is to be performed is equal to m $\times$ n where m = the upper limit of the E switch and n = the upper limit of the F switch. (For example, 65 = 5 $\times$ 13.)

When programming for the 220-word memory E101, the band switch can also be used for counting. When used in conjunction with the E and F switches, the three switches can count up to 4 $\times$ 256, or 1024.

3. Split register storage

It is sometimes desirable to use split register storage—i.e., to store two or more items in each memory location. One reason for using split register storage is to keep related information such as code number and quantity together. The four digit positions on the right might be used for code number and the other eight digit positions for quantity. Another important reason for using split register storage is to increase the capacity of the memory. A table of 180 6-digit rates or 270 4-digit rates can be stored in 90 memory locations. The only restriction is that all items stored in one location be of the same sign. This does not hold true if the quantities are always of opposite sign or if only one item has a sign and others (such as code number) are neither positive or negative. Another way of keeping track of the sign is to code it in as part of the number—"0," for example, standing for "plus" and "1" for "minus."

When more than one number is stored in a memory location, the programmer extracts the number he wants by using shift instructions. To illustrate, consider the case where 3 4-digit rates are stored in location 99. Assuming rate 1 is the one on the left, rate 2 is the one in the center, and rate 3 is the one on the right, the programmer would call for them in the program as follows:

99 | 111122223333 |

Rate 1	Rate 2	Rate 3
R 9 9	R 9 9	R 9 9
A 2 8	A 1 4	A 1 8
	A 2 8	A 2 8

In each case the rate selected would appear all the way to the right of the accumulator, with zeros in the other 8 digit positions. The original 12-digit number would remain in the memory unchanged.

4. Split register storage used in conjunction with X and Y keys for random access.

The X and Y keys on the left side of the keyboard are frequently used when random access is made to the memory. The programmer refers to memory location X Y in the program and has the operator depress the proper X and Y keys when running the problem.

When split register storage is used in conjunction with random access, the programmer must make provision for selecting not only the proper memory location but also the proper *part* of the memory location in which the particular number he wants is stored. He might well be faced with a problem involving 90 memory locations, each storing 3 4-digit rates. One solution to this problem is to enter a digit into the keyboard which designates the position of the rate ("1" for left, "2" for center, and "3" for right) at the time the X and Y keys designating the proper memory location are depressed. Then, by using the "C" instruction combined with a few subtraction instructions, the proper rate can be selected.

To use this approach, a "2" and a "1" are stored in memory locations 99 and 98 respectively. The first instruction in the routine is a "K" instruction, at which point the operator depresses the proper X and Y keys and enters a 1, 2, or 3 into the keyboard depending on whether he wants rate 1 (on the left), rate 2 (in the center), or rate 3 (on the right). The next instruction is −99. It subtracts the "2" stored in 99 from the 1, 2, or 3 just entered through the keyboard. The answer will be positive if a 2 or 3 was entered but negative if a 1 (indicating the rate at the left) was entered. This enables the programmer to extract rate 1 by means of a "C" instruction. In order to extract rate 2 or 3, there might be another subtraction instruction, −98, followed by another "C" instruction. The program for this routine is as follows:

0	K	Set X and Y. Enter a 1, 2 or 3 for rates "1," "2," or "3" respectively.
1	− 9 9	If rate 1, acc. will be neg. (1 − 2 = −1) If rates 2 or 3, acc. will be pos. (2 − 2 = 0, 3 − 2 = 1)
2	C 0 11	Transfer to step 11 if rate 1. Continue with next step if rates 2 or 3
3	− 9 8	If rate 2, acc. will be neg. (0 − 1 = −1) If rate 3, acc. will still be pos. (1 − 1 = 0)
4	C 0 8	Transfer to step 8 if rate 2. Continue with next step if rate 3
5	R X Y	Rate 3—Shift to left; transfer to step 12
6	A 1 8	
7	U 0 12	
8	R X Y	Rate 2—Shift to left; transfer to step 12
9	A 1 4	
10	U 0 12	
11	R X Y	Rate 1—Already at left; continue with step 12
12	A 2 8	Shift all the way to right, leaving zeros in the other 8 digit positions.
13	_ _ _	

5. Equality Test

This is a simple routine for determining whether two numbers are equal. It is based on the fact that the E101 always considers zero as positive when making a conditional transfer. The equality test consists of three instructions: the first step subtracts one of the two numbers from the other (the order is unimportant); the second instruction is "A 4" which makes the difference negative; the third instruction is a "C." If the difference between the two numbers is anything but zero, it will be negative as a result of the A4 instruction and will cause the E101 to transfer to another part of the problem. If the difference between the two numbers is zero, indicating they are equal, the E101 will not transfer on the "C" instruction but will go on to the next instruction in the program.

This technique of using A 4 and C to distinguish between zero and non-zero numbers in the accumulator could have been employed in the coded extraction routine above. Instruction 3 could have been A4, thus saving memory location 98.

C. SPECIAL INFORMATION ON BASIC INSTRUCTIONS

The ideas included in this section are as follows:

1. Use of last instruction on pinboard to go back to beginning of pinboard.
2. Iterating within a pinboard where possible.
3. Uses of X and Y other than address modification.
4. Subroutine exit.
5. Setting one switch from the other.

1. Use of last instruction on pinboard to go back to beginning of pinboard

The usual way of going from one pinboard to another is to have the last step on each pinboard a

"U" instruction. If step 15 is the last step used and is *not* a transfer instruction, the E101 automatically goes back to the beginning of the same pinboard for its next instruction. This is because each pinboard is scanned by a stepping switch. The position after 15 is 0, just as it is on the E and F switches.

There are times when this can be used to advantage. For example, in a successive approximation routine to obtain the square root of a number, one of the available programs uses exactly 16 steps, filling one complete pinboard. The last step on the pinboard is a "C" instruction. If the approximation to the square root is sufficiently close at that point, the number in the accumulator is negative, and the C _ _ is pinned to go to another pinboard. If, however, (as is usually the case in the first few iterations) the approximation is not sufficiently close, the number in the accumulator is positive, and the E101 goes directly from the "C" instruction pinned in step 15 to the next instruction, step 0, of the same pinboard for another iteration, which will bring the approximation closer to the actual square root.

2. Iterating within a pinboard where possible

Generally, it takes less time to transfer to a step in the same pinboard than to a step in a different pinboard. When the transfer is to the next adjacent pinboard, the time involved is slight. The slowest transfer is from a given pinboard to the one preceding it. For example, to transfer from pinboard 4 to pinboard 3, the machine's pinboard control stepping switch must step from position 4 to 5 to 6 to 7 to 8 to 1 to 2, and then to 3. If a transfer such as this occurs only a few times in a problem, the time involved is negligible.

If, however, it is part of an iterative routine that is repeated many times in the course of a problem, the milliseconds can add up to valuable minutes. For example, consider a 13-step iterative loop starting in pinboard 5 and ending in pinboard 6, that is repeated ninety or more times. The transfer from pinboard 5 to pinboard 6 takes little time, but each time the iteration is repeated, it is necessary to return from pinboard 6 to pinboard 5 for the next iteration. This involves stepping the pinboard control switch from position 6 to 7, 8, 1, 2, 3, 4, and finally to 5—not just once or twice, but ninety or more times. Since there are only 13 steps in the iteration, it would be better in this case to include all 13 steps in a single pinboard, cutting out unnecessary transfers to other pinboards and reducing running time. There are many problems like this one where running time on the E101 can be substantially reduced with a little forethought in programming.

3. Uses of X and Y other than address modification

Although the X and Y keys are generally used for manual address modification, either as a supplement to E and F, or as a means of random access, they are not limited to this function. X may be pinned in the second area of the pinboard and Y in the third area in *any* instruction. Some specific cases where they have been used to advantage are as follows:

a. *Dispatching*—U X Y—Using X and Y to denote pinboard and step number, respectively, in a transfer instruction, thus allowing the transfer to be made to any step (between 0 and 9) on any pinboard by depressing the proper X and Y keys.

b. *Shifting*—A X 4, A 2 Y, etc.—Pinning X instead of "1" for left or "2" for right enabling the operator to select direction of shift (helpful when used with split register storage). Also, programming Y instead of a specific number of places to be shifted (helpful when scaling a problem with numbers that vary in size over a wide range).

c. *Printing*—P X _, P 3 Y, etc.—Using X to designate motor bar number, allowing operator to exercise a greater control over format. Setting Y = 0 allows operator to suppress printing.

d. *Changing Limit of "S" Instructions*—S O Y, etc. —Pinning Y instead of a definite limit in any S instruction when working with an indefinite or variable number of amounts.

In addition to the above examples, X has also been used in special cases in the second area of "S" and "H" instructions.

4. Subroutine Exit

The instruction, "U a *," where "a" is any number from 1 to 8, sends the E101 to the step on pinboard "a" immediately following the last step performed on that pinboard. For example, if the E101 transfers from pinboard 3, step 6, to a subroutine in pinboard 5 where the last step is U 3 *, at the end of the subroutine the E101 will return to pinboard 3 and go on with step 7.

This technique works because of the nature of pinboard scanning: recall that each pinboard is scanned by its own stepping switch. The execution of an instruction in a pinboard automatically steps that pinboard switch to the following instruction. (This is true even if the instruction executed is a transfer to some other pinboard.) Thus, *each of the 8 pinboard switches is always set at the step following the step last executed in that pinboard.* Pinning a "*" in the third area of a transfer instruction simply uses the present setting of the pinboard switch selected in the second area.

By modifying this instruction somewhat, making

it "U E *," greater flexibility can be obtained. It is very useful where, after transferring to a given subroutine from several different pinboards in a problem, one must return in each case to the pinboard from which the transfer was made. Before leaving each pinboard to go into the subroutine, the E switch is set to agree with the number of that particular pinboard (e.g., in pinboard 3, H 0 3). The last step in the subroutine is "U E *," returning the E101 to the step following the last step executed on pinboard "E," where E was set to designate the pinboard from which the last transfer was made.

5. Setting one switch from the other

There are times when it is helpful to set one switch to the setting of the other using the instruction "H 1 E" or "H 0 F." One place where "H 1 E" can be used to advantage is in developing a triangular matrix of squares and cross-products with the squares along the diagonal. Assuming there are nine variables designated as $x_0, x_1, \ldots . x_8$ stored in memory locations 00 to 08 respectively, their squares and cross-products can be computed and stored in the following array with the use of the "H 1 E" instruction:

	0	1	2	3	4	5	6	7	8	9
0	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	
1	X_0^2	X_0X_1	X_0X_2						X_0X_8	
2		X_1^2	X_1X_2						X_1X_8	
3			X_2^2	X_2X_3					X_2X_8	
4				X_3^2	X_3X_4				X_3X_8	
5					X_4^2	X_4X_5			X_4X_8	
6						X_5^2	X_5X_6		X_5X_8	
7							X_6^2	X_6X_7	X_6X_8	
8								X_7^2	X_7X_8	
9									X_8^2	

By homing F to E at the beginning of each row, each cross-product is developed only once, thus saving both running time and memory space. The program for this routine shown at the right.

0	H	1	0	
1	H	0	1	
2	R	0	F	
3	B			
4	×	0	F	
5	W	E	F	
6	S	1	8	
7	U	0	4	
8	H	1	E	
9	S	0	9	
10	U	0	2	

In this particular problem it works out well to store the matrix in upper right triangular form. In some problems it is more convenient to store the matrix in upper left triangular form or lower left triangular form.

D. ALGEBRAIC MANIPULATIONS

The ideas included in this section are as follows:

1. Polynomial evaluation.
2. Forced iteration.
3. Programming a problem in linear fashion to save running time.
4. Multi-purpose subroutine.
5. Angle reduction.
6. Matrix structure.

1. Polynomial evaluation

There is a special type of factoring used in evaluating polynomials that is considered almost standard with electronic digital computers. Sometimes referred to as "synthetic division," it permits the computer to evaluate the polynomial in a minimum of program steps consisting of successive multiplications and additions.

To illustrate, consider the series approximation for e :

$$e = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!} + \frac{x^8}{8!} + \cdots$$

When carried out to the 9th term, this problem can be programmed for the E101 in only 10 program steps provided the special type of factoring is used.

The first step is to multiply the coefficient of the highest power of x by x:

$$\frac{1}{8!}\,x.$$

Then the coefficient of the second highest power of x is added and the sum again multiplied by x:

$$\left(\frac{1}{8!}x + \frac{1}{7!}\right)x = \frac{x}{7!} + \frac{x^2}{8!}.$$

Next, the coefficient of the third highest power of x is added and the sum again multiplied by x:

$$\left(\frac{x}{7!} + \frac{x^2}{8!} + \frac{1}{6!}\right)x = \frac{x}{6!} + \frac{x^2}{7!} + \frac{x^3}{8!}.$$

The process is continued, each time adding the coefficient of the next power of x and multiplying the sum by x. At the end of the iteration, the polynomial is in the form:

$$x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!} + \frac{x^8}{8!}.$$

The only remaining step is to add "one."

The same basic approach is used to evaluate a polynomial such as the truncated series approximation for cosine x:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Instead of multiplying by x each time, x^2 is used.

2. Forced Iteration

In most cases it is rather obvious when to iterate in a problem. For instance, no programmer would consider stretching out in linear fashion the program for loading 60 amounts into the memory. By a simple iterative process the loading can be programmed in just 8 steps while the same operation in a non-iterative, linear fashion requires 120 steps (not counting transfers between pinboards).

Sometimes the situation is not so obvious. At times a problem with seemingly no apparent pattern of repitition can be forced into iterative form to save program steps. One problem where a complete pinboard was saved by using this technique contained the following summations.

$$\Sigma\, x\, y\, \Delta\, x$$

$$-\frac{1}{2}\,\Sigma\, y^2\, \Delta\, x$$

$$\frac{1}{2}\,\Sigma\, x\, y^2\, \Delta\, x$$

$$-\frac{1}{3}\,\Sigma\, y^3\, \Delta\, x$$

$$\Sigma\, x^2\, y\, \Delta x$$

For each set of entry values of x, y, *and* $\Delta\, x$, $y\, \Delta\, x$ was stored and computed in the B register. x, y, $x\, y$, y^2 and x^2 were stored in memory locations 00 to 04 respectively. By setting up an iterative loop multiplying the contents of 0 F by the contents of the B register, summing and storing in 9 F, the summations were obtained in only a few program steps. After all values of x, y, and $\Delta\, x$ had been entered, another iterative loop multiplied the contents of 9 F (the summations) by the corresponding coefficients: 1, $-\frac{1}{2}$, $\frac{1}{2}$, $-\frac{1}{3}$ and 1 stored in 10, 11, 12, 13 and 14 respectively. In this particular case the complete problem (of which this is only a small part) was programmed in 8 pinboards. This would not have been possible without forced iteration.

3. Programming in a linear fashion to save running time

Sometimes the opposite approach is called for—programming a repetitive type problem in linear fashion instead of iterating. This approach saves running time since there are fewer operations for the E101 to perform even though there are more program steps. The cosine routine when programmed in an iterative loop uses 12 steps and takes 4 seconds while the same routine in linear fashion uses 18 steps but takes less than 3 seconds running time. Sometimes it pays to compromise—iterate part of a routine but not all of it. For example, in clearing the entire memory, there are three basic ways of programming the problem: stretching out the entire routine linearly without using either switch, stretching out part of the routine using one of the two switches, and iterating the entire routine using both the E and F switches. The first approach requires 200 steps and can hardly be considered seriously. The second approach uses 15 program steps and takes about 8 or 9 seconds of running time. The third approach, presented in Part II as the standard one, uses only 9 steps but takes about 80 seconds on the E101. The last two routines appear below for comparison.

Clearing
Semi-iterative
15 Steps, 8 or 9 Seconds

0	R 9 9
1	− 9 9
2	H 1 0
3	W 0 F
4	W 1 F
5	W 2 F
6	W 3 F
7	W 4 F
8	W 5 F
9	W 6 F
10	W 7 F
11	W 8 F
12	W 9 F
13	S 1 9
14	U 0 3

Clearing
Iterative
9 Steps, Approx. 80 Secs.

0	R 9 9
1	− 9 9
2	H 0 0
3	H 1 0
4	W E F
5	S 1 9
6	U 0 4
7	S 0 9
8	U 0 3

4. Multi-purpose subroutine

Since many mathematical routines are based on approximating polynomials, it is fairly easy to incorporate the programs for several of these routines into one multi-purpose subroutine. Since the coefficients and sometimes the structure of the polynomial itself are likely to differ from one polynomial to another, both these factors must be taken into consideration in programming any multi-purpose subroutine.

The variance in coefficients can be taken care of by storing the coefficients for each polynomial in a different row or column of the memory and having the E101 operate on the appropriate row or column.

The variance in polynomial structure (i.e., where one is a polynomial in x, one is a polynomial in x^2,

etc.) can be taken care of by using a forced iteration technique. The multi-purpose subroutine would actually be programmed for a polynomial in x with all terms present: x, x^2, x^3, x^4, etc. When used to evaluate a polynomial in x^2, zeros would be used as the coefficients of the missing terms.

An example of a multi-purpose subroutine that can be used to approximate sine, cosine, arc tangent and e^{-u} is given in the *Handbook of Subroutines and Subroutine Methods for the* ELECTRODATA *101*. In the example given in the subroutine manual, the multi-purpose subroutine is pinned in pinboard 5. In pinboard 1 there is a transfer to the subroutine for a sine approximation; in pinboard 2 there is a transfer to the subroutine for a cosine approximation; in pinboard 3 a transfer for an arc tangent approximation; and in pinboard 4 a transfer for an e^{-u} approximation.

In each case the program calls for returning to the original pinboard after each subroutine. The last step in the subroutine therefore is "U E *," and the step preceding the transfer instruction in each of the four pinboards is "H 0 b" where "b" is the number of the pinboard.

If the coefficients to be used in each approximation can be stored in a row of the memory whose number corresponds to the pinboard number from which the transfer to the pinboard is made, the E switch can perform a two-fold purpose: *it can select the appropriate coefficients to be used in the subroutine, and also select the pinboard to be used in the subroutine exit instruction "U E *."* The F switch is then free for iterating within the subroutine. When programmed in this fashion, only 10 steps are required for the subroutine.

5. Angle reduction

In order to arrive at a close approximation to the sine and cosine of an angle on the E101 in a minimum length of time, the angle must be within a certain range. The programs given in *Handbook of Subroutines and Subroutine Methods for the* ELECTRODATA *101* limit the range to between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.

In addition to the standard subroutines, there are also included in the manual a sine and cosine subroutine for large angles (up to 100 radians). The "large angle" routines make use of a quick method of angle reduction that "reduces" the size of the angle before determining the sine or cosine. The procedure is quite simple and requires only a few more steps than the standard routines: the angle is first divided by 2π and the integer portion of the quotient shifted off. Then the sine or cosine of the remaining decimal portion is determined using the same basic method as used in the standard routines. Only the constants differ.

The method is based on the trigonometric identity:

$$\sin A = \sin 2\pi y \text{ where } A = 2\pi N + 2\pi y,$$

where N is an integer.

If $2\pi y$ is substituted for A in the sine series

$$\sin A = A - \frac{1}{3!}A^3 + \frac{1}{5!}A^5 - \frac{1}{7!}A^7 + \ldots,$$

we have:

$$\sin A = (2\pi\ y - \left[\frac{(2\pi)^3}{3!}\right] y^3 + \left[\frac{(2\pi)^5}{5!}\right] y^5 - \left[\frac{(2\pi)^7}{7!}\right] y^7 + \ldots.$$

If we let $a_1 = (2\pi)$

$$a_3 = -\frac{(2\pi)^3}{3!}$$

$$a_5 = \frac{(2\pi)^5}{5!}, \text{ etc.}$$

then $\sin A = a_1 y + a_3 y^3 + a_5 y^5 + a_7 y^7 + \ldots$.

To arrive at y, we divide A by 2π and shift off the integer portion of the answer:

If $A = 2\pi N + 2\pi y$,

then $\frac{A}{2\pi} = N + y$

With the integer portion, N, shifted off, the remaining decimal portion is equal to y. Substituting y in the above sine series, we arrive at the approximation for sine A. The cosine approximation uses the same basic approach.

NOTE: In the form in which this routine is given in *Handbook of Subroutines and Subroutine Methods for the* ELECTRODATA *101*, the constants a_1, a_3, etc. include an expansion factor to improve the accuracy of the approximation.

6. Matrix structure

In the field of matrix algebra, there are E101 programs for matrix inversion, solution of simultaneous equations, computations of eigenvalues, etc. These programs are written up as separate E101 pamphlets and are available from the ELECTRODATA Division Applied Mathematics Department. It seems worthwhile calling attention here to how the E and F switches of the E101 permit a kind of programming of matrix calculations that more naturally follows the mathematical character of matrix algebra than one generally finds in other computers.

For example, consider the simple problem where an $N \times N$ matrix is multiplied by a column vector. A common way of formally expressing this operation is as follows:

$$P_i = \sum_{j=1}^{j=N} (A_{ij} \cdot B_j) \qquad i = 1,2,\cdots N$$

where:

P_i is the product column vector
A_{ij} is the matrix, and
B_j is the multiplier column vector

Essentially the character of the E101 program requires only the substitution of "E" and "F," representing the E101 switches, for the subscripts *"i"* and *"j"*:

$$P_E = \sum_{F=1}^{F=Y} (A_{EF} \cdot B_F)$$

Assuming that *B* is stored in *row* zero of the memory and that each element of the matrix is stored in the memory location corresponding to its subscript (for example, A_{35} in location 35), the problem can be programmed as follows:

0	H	0	1	Home E to 1
1	H	1	1	Home F to 1
2	R	E	0	Clear a location for P_E
3	−	E	0	
4	W	E	0	
5	R	E	F	A_{EF}
6	B			
7	×	0	F	$\Sigma (A_{EF} \cdot B_F)$
8	+	E	0	
9	W	E	0	
10	S	1	Y	Step F once (limit of F is Y)
11	U	0	5	
12	P	3		
13	S	0	Y	Step E once (limit of E is Y)
14	U	0	1	
15	A		*	Halt: Problem finished. P_E has been printed and stored in column zero.

NOTE: As programmed above, the Y keys are used to indicate the order of the matrix. An alternative method is to pin the value of *N* in the 3rd area of steps 10 and 13, and change the pin whenever the program is used for a matrix of a different order.

APPENDIX

PRACTICE PROBLEM SOLUTIONS

1. a. P.B. No. 1

```
 0   K
 1   W 7 0
 2   K
 3   W 7 1
 4   K
 5   W 7 2
 6   K
 7   W 7 3
 8   K
 9   W 7 4
10   K
11   W 7 5
12   K
13   W 7 6
14   K
15   U 2 0
```

P.B. No. 2

```
0   W 7 7
1   K
2   W 7 8
3   K
4   W 7 9
5   A _ *
```

b. P.B. No. 1

```
 0   H 1 0
 1   K
 2   W 7 F
 3   S 1 9
 4   U 0 1
 5   A _ *
```

c. P.B. No. 1

```
 0   H 0 0
 1   K
 2   W 7 E
 3   S 0 9
 4   U 0 1
 5   A _ *
```

2. a. P.B. No. 1

```
 0   K
 1   B
 2   K
 3   W 0 0
 4   ÷ 2 0
 5   K
 6   W 0 1
 7   ÷ 2 1
 8   K
 9   W 0 2
10   ÷ 2 2
11   K
12   W 0 3
13   ÷ 2 3
14   K
15   U 2 0
```

P.B. N. 2

```
 0   W 0 4
 1   ÷ 2 4
 2   K
 3   W 0 5
 4   ÷ 2 5
 5   K
 6   W 0 6
 7   ÷ 2 6
 8   K
 9   W 0 7
10   ÷ 2 7
11   K
12   W 0 8
13   ÷ 2 8
14   K
15   U 3 0
```

P.B. No. 3

```
0   W 0 9
1   ÷ 2 9
2   A _ *
```

b. P.B. No. 1

```
 0   K
 1   B
 2   H 1 0
 3   K
 4   W 0 F
 5   ÷ 2 F
 6   S 1 9
 7   U 0 3
 8   A _ *
```

3. P.B. No. 1

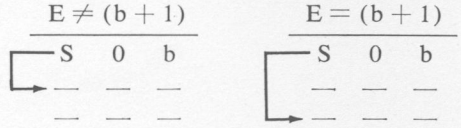
```
 0   R 9 9
 1   — 9 9      Clear mem. loc. 99
 2   W 9 9
 3   H 1 0
 4   K
 5   W 8 F
 6   + 9 9
 7   W 9 9
 8   S 1 9
 9   U 0 4
10   A _ *
```

4. P.B. No. 1

```
 0   R 9 9
 1   — 9 9      Clear acc.
 2   H 0 0
 3   H 1 0
 4   W E F
 5   S 1 9
 6   U 0 4
 7   S 0 3
 8   U 0 3
 9   A _ *
```

ELECTRODATA 101 INSTRUCTIONS

PINBOARD AREA 1	2	3	Operation Carried Out
+	a	b	Add the contents of memory location ab to the contents of the accumulator. (a = tens digit of the memory address, b = units digit of the memory address.)
—	a	b	Subtract contents of ab from contents of accumulator.
×	a	b	Multiply the contents of the B register by the contents of memory location ab (answer is in the accumulator).
÷	a	b	Divide the contents of accumulator by the number in the B register and store the answer in memory location ab (remainder left in accumulator).
R	a	b	Read the contents of memory location ab into the accumulator (leaving copy in ab).
W	a	b	Write the contents of the accumulator into memory location ab, leaving copy in accumulator.
A	1	b	Shift the contents of the accumulator b places to the left. (b ≦ 10).
A	2	b	Shift the contents of the accumulator b places to the right. (b ≦ 10).
A	3	—	Make the contents of the accumulator positive.
A	4	—	Make the contents of the accumulator negative.
A	5	—	Change the sign of the contents of the accumulator.
A	—	*	Halt the machine.
B	—	—	Transfer the contents of the accumulator into the B register, leaving copy in accumulator.
K	—	—	Transfer the contents of the keyboard into the accumulator when operator depresses a motor bar.[1]
P	a	—	Print contents of the accumulator using Motor Bar "a", leaving copy in accumulator.[2]
U	a	b	Unconditionally transfer to pinboard a, step b.[3]
C	a	b	Conditionally transfer to pinboard a, step b; i.e., if the contents of the accumulator is negative execute the transfer; if the contents of the accumulator is positive execute the next step in the program.[3]
S	0	b	Step the E switch once; then if E ≠ (b + 1), execute the next instruction; if E = (b + 1), execute the instruction *after* the next instruction.



S	1	b	Same as above, but using the F switch.
S	2	b	Step E and F switches once; then if E ≠ b + 1, execute the next instruction; if E = b + 1, execute the instruction after the next instruction.

PINBOARD AREA 1	2	3	Operation Carried out
H	0	b	Home the E switch, stopping at position b.
H	1	b	Home the F switch, stopping at position b.
H	2	b	Home the E switch, stopping at position b, and advance the F switch the same number of places as E moves.

Pinning E instead of a definite number for a, or E or F instead of b, sends the machine to the appropriate switch's setting for that particular value of a or b.

Pinning X for a, or Y for b, sends the machine to the appropriate keyboard setting of the X or Y keys for that particular value of a or b.

NOTES: [1]Pinning the 0 in level 3 affects a non-print for this operation.
[2]Pinning the 0 in level 3 gives a non-print, while the * gives a print-and-halt.
[3]Pinning 0 in level 2 keeps the transfer (to instruction b) in the same pinboard, while the * in level 3 transfers to whatever instruction was last executed (in pinboard a) + 1.

TAPE INPUT UNIT INSTRUCTIONS

T	—	11	Tape Transfer—Execute the next instruction on Tape. Follow each instruction on tape in sequence until control is returned to pinboard by a "U" or "C" instruction on tape.
T	—	12	Tape Read—Read the next number on tape into the accumulator and then continue with the next pinboard instruction.

All instructions on tape require 3 characters, e.g., the B instruction on tape would be used as B 1 1. Numbers are of variable word length, 1-12 digits starting with the most significant digit. It is therefore necessary to have a begin word character and an end word character. The minus sign should precede the digits when present.

220-WORD MEMORY INSTRUCTIONS

H	3	b	Home the M switch (expanded memory band selector switch) to b. b may have only the values 0, 1, 2 or 3.
S	3	b	Step — Step the M switch once; then if M ≠ b + 1, execute the next instruction; if M = b + 1, execute the instruction after the next instruction.
H	4	—	Increase the E switch setting by the number in the least significant digit position of the accumulator.
H	5	—	Same using F switch.
H	6	—	Same using E and F.
H	7	—	Same using the M switch.

V SWITCH INSTRUCTIONS

In addition to the instructions for the basic computer:

Pinning "V" instead of a definite number for b in the 3rd area of the pinboard sends the machine to the appropriate setting for "V" as determined by carriage position.

PUNCHED TAPE OUTPUT INSTRUCTIONS

PUNCHING DATA

PINBOARD AREA 1	2	3	
P	0	0	**Punch**—Punch data stored in the accumulator with a non-tab, non-print, and non-space of the Keyboard-Printer.
P	5	—	**Print and Punch**—Print the contents of the accumulator, perform Motor Bar 1 carriage positioning function after printing, and punch the data on tape.[1]
P	6	—	**Print and Punch**—Same as P 5 except perform Motor Bar 2 carriage positioning function.[1]
P	7	—	**Print and Punch**—Same as P 5 except perform Motor Bar 3 carriage positioning function.[1]
P	8	—	**Print and Punch**—Same as P 5 except perform Motor Bar 4 carriage positioning function.[1]
K	0	0	**Keyboard Punch**—Halt the program and light the keyboard signal. When the operator touches a motor bar, transfer the number in the keyboard to the 11 least significant digits of the accumulator, and punch the 12-digit contents of the accumulator into the tape.
K	5	—	**Keyboard Print and Punch**—Halt the program and light the KEYBOARD signal. When the operator touches a motor bar: (1) the number in the keyboard prints and is transferred to the 11 least significant digits of the accumulator; (2) the 12-digit number in the accumulator is punched into the tape; (3) and the carriage moves in accordance with the motor bar action selected by the operator.

NOTES: 1. a. Pinning a 0 in the third level effects a non-print for this operation.
b. Pinning a * in the third level effects a computer HALT after the print and punch operations are completed.
2. Pinning a 0 in the third level effects a non-print for this operation.

PUNCHING INSTRUCTIONS

The Tape Output Unit is told to punch E101 instructions into tape by means of an Mab instruction. The "b" pin setting selects the operation code to be punched, the E switch setting selects the code to be punched in the tens level, and the F switch setting selects the code to be punched into the units level.

As is the case when programs are punched into tape on other tape preparation devices, each tape instruction must consist of 3 characters. Filler digits (ones) must be used in instructions with one or two characters (e.g., B11). The punch instruction operation is coded as follows:

PINBOARD AREA 1	2	3
M	0	b

Punch Instruction—Punch the 3-character coded computer instruction designated as follows:

OPERATION

"b" Pin Setting	Operation Punched (coded)
0	+
1	−
2	×
3	÷
4	R
5	W
6	A
7	B
8	K
9	P
10	U
11	C
12	S
13	H
14	M
15	delete

TENS AREA

E Switch Setting	Tens Level Character Punched
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	E
11	tab
12	X
13	carriage-return
14	stop
15	delete

	UNITS AREA
F Switch Setting	Units Level Character Punched
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15

PINBOARD AREA		
1	2	3
M	1	b

Punch Instruction—Same as M 0 b in operation and tens level character punched. Same as M 0 b in units level punched except as follows:

	UNITS AREA
F Switch Setting	Units Level Character Punched
10	E
11	F
12	Y
13	*
14	V
15	delete

ElectroData
DIVISION OF BURROUGHS CORPORATION

460 SIERRA MADRE VILLA,
PASADENA, CALIFORNIA

DISTRICT OFFICES

BOSTON
NEW YORK
PHILADELPHIA
ROCHESTER
WASHINGTON
CHICAGO
DALLAS
DETROIT
LOS ANGELES
SAN FRANCISCO
SEATTLE
OTTAWA
MONTREAL
TORONTO

Printed in U.S.A.