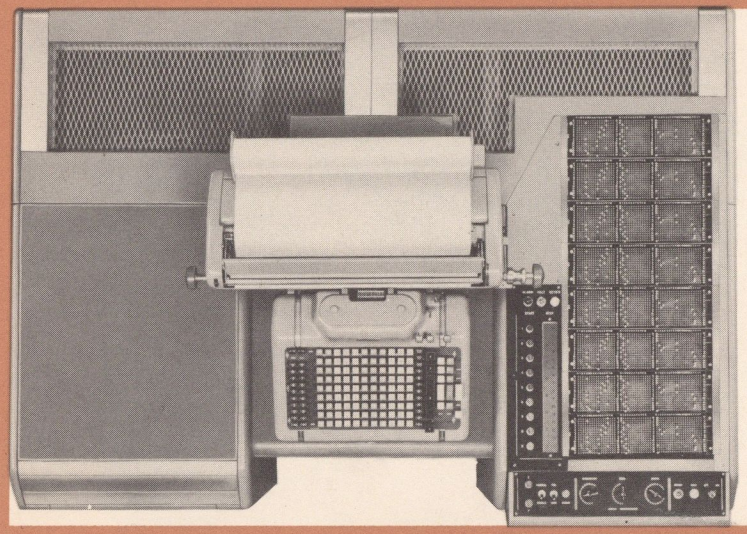
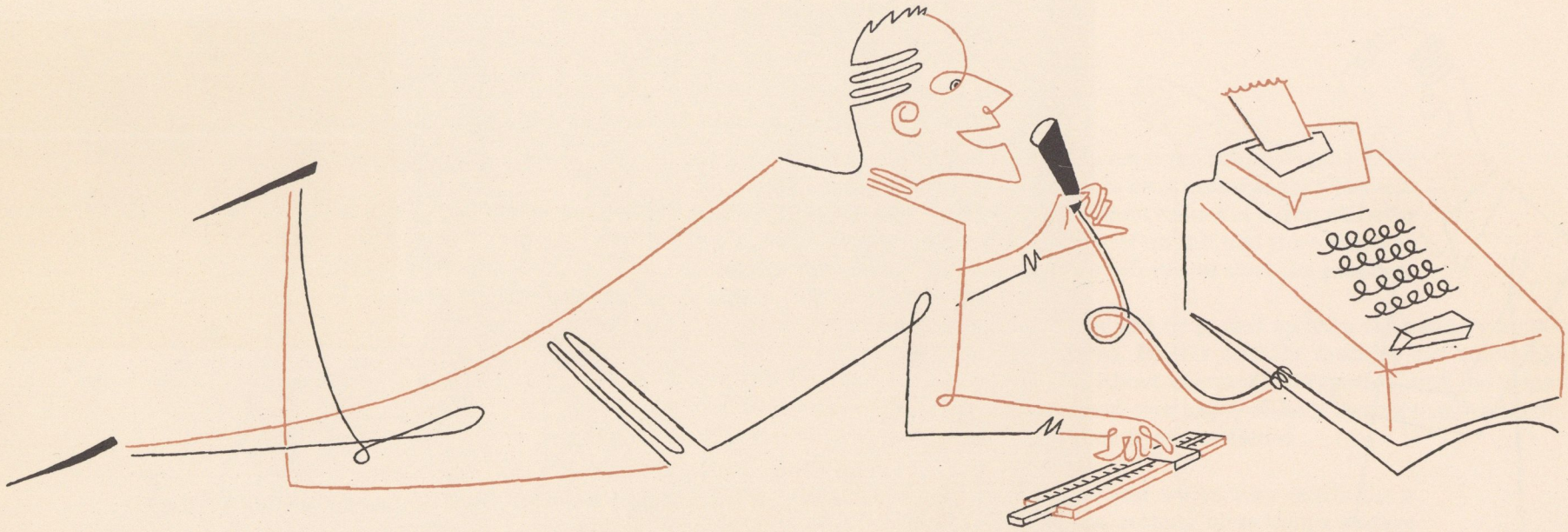


You can program the Burroughs **E 101**

COPYRIGHT 1955

BURROUGHS
CORPORATION

PRINTED IN THE UNITED STATES OF AMERICA

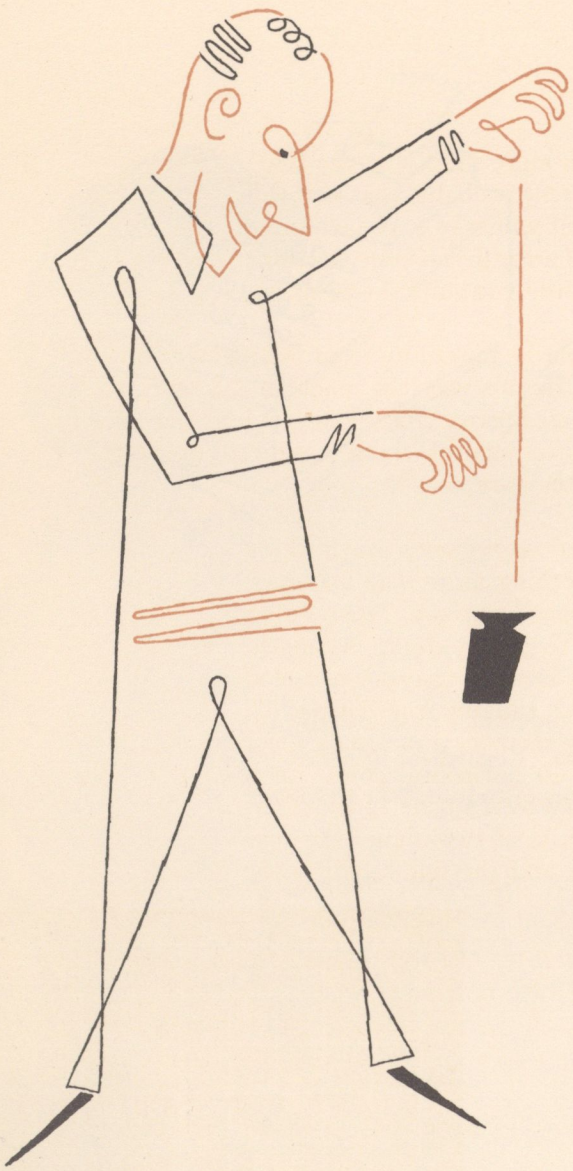


YOU can program the Burroughs E101. It's EASY! You don't need an advanced degree in mathematics or electronics. All you need are a few facts about the E101 and some good common sense!

Have you ever programmed before? You might not realize it, but you probably have. Because programming is nothing more than telling an operator *or a machine* how to solve a problem. If you have ever worked out a procedure for doing calculations of any type, then you have programmed! It might have been for an adding machine, slide rule, desk calculator or bookkeeping machine. Regardless of the machine, if you worked out the procedure, you were programming.

Of course each machine has its own characteristics and has to be programmed accordingly. Some machines print, for example, while others do not. A few features of the Burroughs E101 to remember when programming are:

1. It prints . . . on any type of document.
2. It computes completely automatically.
3. Operator can override and use his own judgment.
4. It can remember (more about this later).



A PROBLEM

Now let's suppose we want to solve the following problem:

$$S = S_0 - 1/2\ gt^2$$

where S is the distance from the ground at the end of t seconds of a body at rest falling from a point S_0.

If you were writing instructions for doing this problem on a *desk calculator*, you might proceed as follows:

1. Enter t in Keyboard.
2. Multiply by t to get t^2.
3. Enter 1/2 g in Keyboard.
4. Multiply by t^2 to get $1/2\ gt^2$.
5. Copy result from lower dial (since it must be re-entered later).
6. Enter S_0 in Keyboard.
7. Subtract $1/2\ gt^2$ from S_0.
8. Copy final answer from lower dial to report.

If you were programming the same problem on the *Burroughs E101*, you would use the same instructions as above, but give them directly to the machine since it works automatically (more about *how* these instructions are given later). You would use symbols both you and the machine could readily understand.

Instead of:	*You would write*:
1. Enter t in **K**eyboard	**K**
2. Multiply **(×)** by t to get t^2	×
3. Enter 1/2 g in **K**eyboard	**K**
4. Multiply **(×)** by t^2 to get $1/2\ gt^2$	×
5. Copy (**W**rite) result from lower dial	**W**
6. Enter S_0 in **K**eyboard	**K**
7. Subtract **(−)** $1/2\ gt^2$ from S_0	−
8. Copy final answer from lower dial to report.	**P** (Print)

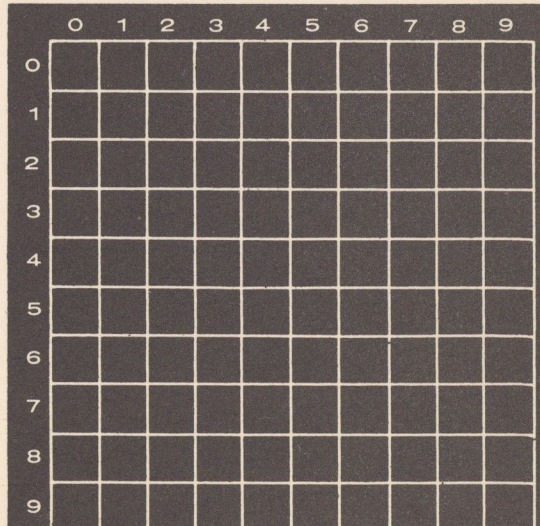
That's how simple it is!

MEMORY

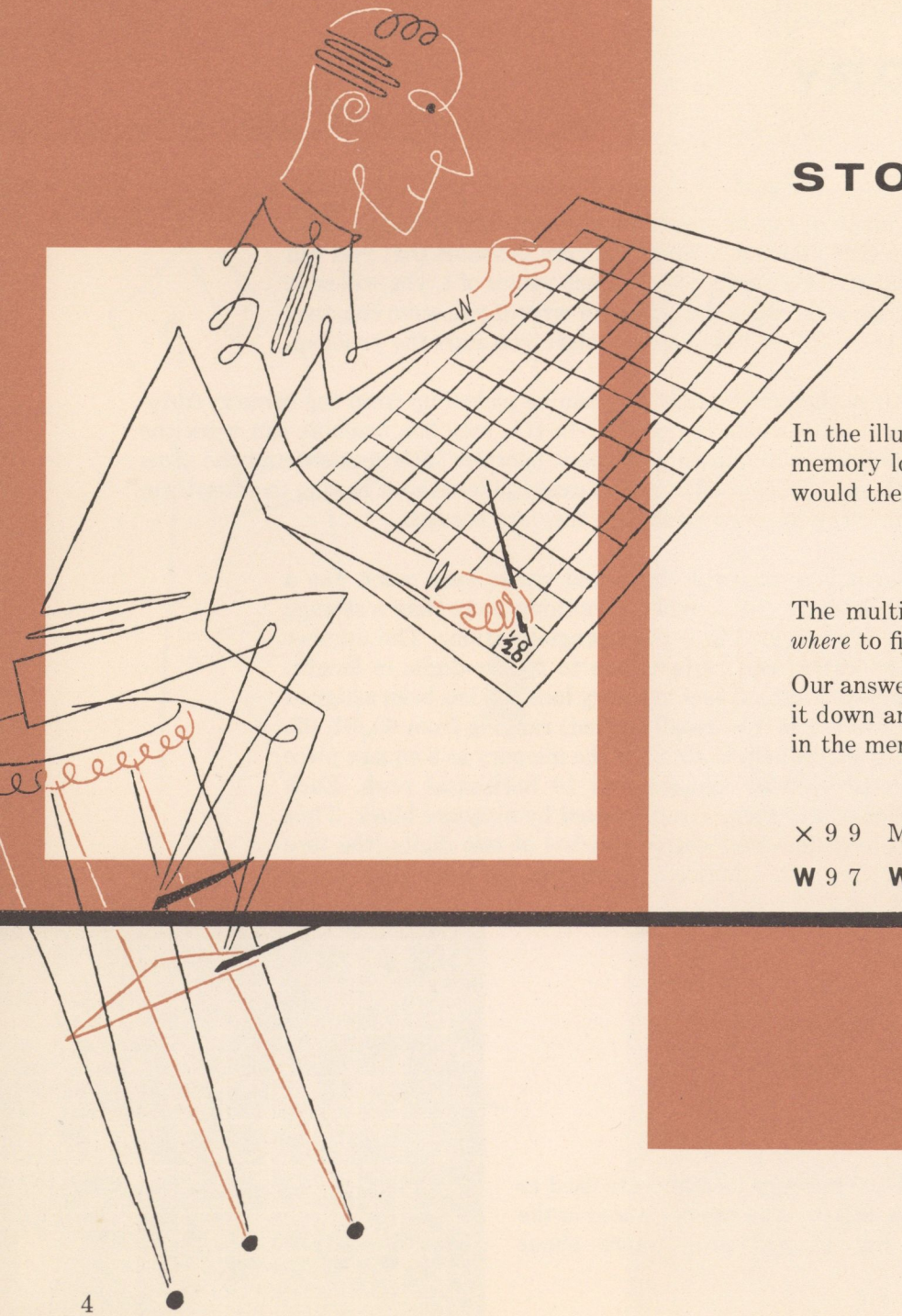
Actually, the problem can be solved much more simply on the E101 by making use of the memory. Since it is very probable that you will want to compute S for a number of different values of t, you will save considerable time and reduce the chance of error if you store constant values like S_0 and 1/2 g in the memory before you start the problem.

Once they are stored, they can be picked up from the memory automatically whenever needed. In this way the machine can work the problem automatically without operator intervention, and the same figures can be used over and over again without having to enter them into the machine each time they are needed.

The memory unit is a magnetic drum and works very much like a tape recorder. Actually the memory is nothing more than a storage place for data. It is divided into 100 storage locations. The number stored in each location can be from one to twelve digits in length. In order to simplify matters, each memory location has been assigned a number (or address as it is usually called) ranging from 00, 01, 02, etc., up to 99. It might help to think of the memory as a square piece of paper with 10 vertical columns and 10 horizontal rows. Each memory location would then be represented by a square block. Thus the address of any memory location consists of two digits: the tens digit is the number of the horizontal row and the units digit is the number of the vertical column.



Note: All 100 memory locations are used to store data. Instructions are *not* stored in the memory but in pinboards (more about this later).



STORAGE

As mentioned before, the memory is for storage only. It is not used for actual computation. All arithmetic operations take place in the accumulator. We merely store amounts in the memory until we are ready to use them to perform some arithmetic computation in the accumulator.

In the illustration shown earlier you might have stored S_0 in memory location 98 and 1/2 g in memory location 99 before starting the problem. Your instruction for multiplying t^2 by 1/2 g would then read:

× 9 9 Multiply t^2 by the number stored in memory location 99 (which is 1/2 g).

The multiplication sign (×) tells the machine *what* to do, and the two numbers (99) tell it *where* to find the amount to work with.

Our answer, 1/2 gt^2, is in the accumulator at this point. Our original instructions call for writing it down and later subtracting it from S_0. Instead of writing it on a piece of paper, let's write it in the memory. If we use memory location 97 for this purpose, our program will read:

× 9 9 Multiply t^2 by 1/2 g stored in memory location 99.

W 9 7 **W**rite the answer now in the accumulator in memory location 97.

SUBTRACTION

Now we are ready to use S_0 which we have previously stored in memory location 98. Our first step is to read it out of memory location 98 into the accumulator, or:

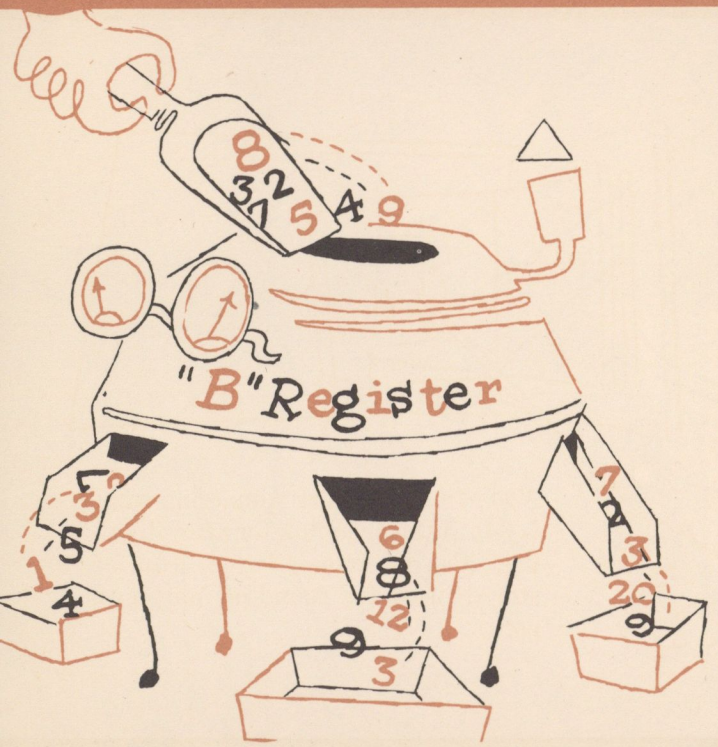
R 9 8 **R**ead contents of memory location 98 into the accumulator.

Next we instruct the E101 to subtract $1/2\ gt^2$ from S_0. Since $1/2\ gt^2$ is now in memory location 97, we write:

– 9 7 Subtract (–) contents of memory location 97 from S_0 already in the accumulator.

The answer in the accumulator is now the distance we are looking for, $S_0 - 1/2\ gt^2$. We are ready to print our answer; so the next step in our program will be:

P **P**rint contents of the accumulator on report.



MULTIPLICATION

When you multiply a series of numbers on a *desk calculator* by a constant multiplier, you store the multiplier in a special register so that you can use it over and over again without the necessity of entering it into the keyboard each time.

The same thing is true of *the E101*. Whenever you multiply or divide on the E101, you store the multiplier or divisor in a special register called the "B" register. A number in the B register behaves just like a number in a memory location: it remains unchanged until you write another number over it. The instruction which does this is:

B Transfer contents of accumulator into **B** register.

A factor once stored in the B register can be used any number of times, and can be changed only by a subsequent B instruction.

In our distance problem we have already shown that to multiply t^2 by 1/2 g which is stored in memory location 99, we write:

× 9 9

Prior to this, however, we must put t^2 into the B register. The same thing is true of t earlier when we multiply t by itself to arrive at t^2.

Our entire problem would be programmed as follows:

K	Enter t in **K**eyboard.
W 9 6	**W**rite t in memory location 96.
B	Transfer t from accumulator into **B** register.
× 9 6	Multiply t in B register by t in 96.
B	Transfer t^2 from accumulator into **B** register.
× 9 9	Multiply t^2 in B register by 1/2 g stored in 99.
W 9 7	**W**rite the answer ($1/2\ gt^2$) now in accumulator in memory location 97.
R 9 8	**R**ead S_0 stored in 98 into accumulator.
− 9 7	Subtract $1/2\ gt^2$ in 97 from S_0 in accumulator.
P	Print answer.

Note: If we wanted to compute t^3 at this point, we would program

× 9 6 Multiply t^2 (already in B register) by t (in memory location 96).

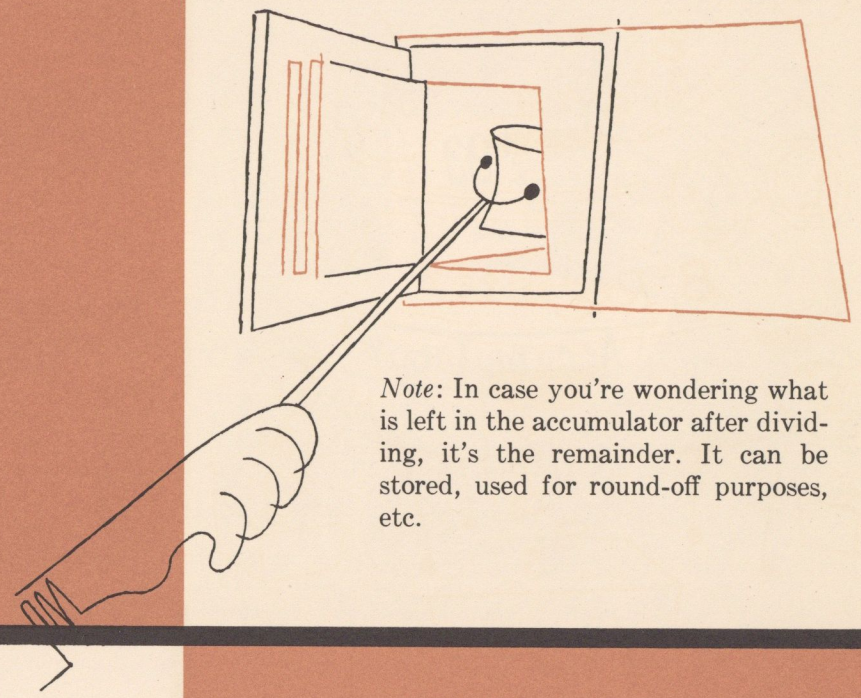
DIVISION

Division is similar to multiplication with two exceptions:

1. Instead of multiplying contents of some *memory location* by what is in the B register, you divide contents of the *accumulator* by what is in the B register.
2. Your answer appears in the *memory* instead of the accumulator as it would in multiplication.

If you had been *dividing* by t^2 in your problem instead of multiplying, the program would have looked like this:

K	Enter t in **K**eyboard.
W 9 6	**W**rite in 96.
B	Put in **B** register.
× 9 6	Multiply t in B register by t in 96.
B	Put t^2 in **B** register.
R 9 9	**R**ead 1/2 g from 99 into accumulator.
÷ 9 5	Divide 1/2 g in accumulator by t^2 in B register. Store answer in 95.



Note: In case you're wondering what is left in the accumulator after dividing, it's the remainder. It can be stored, used for round-off purposes, etc.

ADDITION

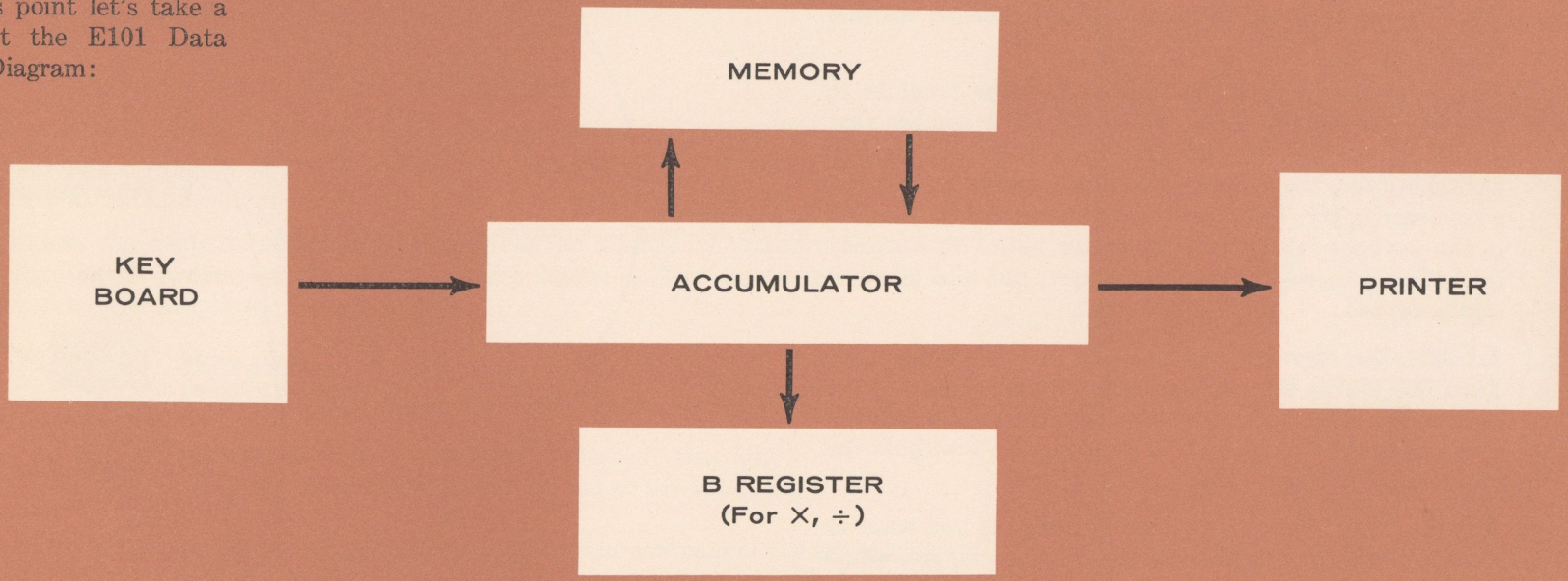
Addition is very similar to subtraction. When you want to add the contents of memory location 93, for example, to the number in the accumulator, your program would read:

+ 93 Add number in location 93 to number already in the accumulator.

Here again the plus sign (+) tells the E101 *what* to do, and the two numbers (93) tell it *where* to find the amount to work with.

E101 DATA FLOW DIAGRAM

At this point let's take a look at the E101 Data Flow Diagram:



A few things that should be remembered are:

1. When you *write* a number from the accumulator into the memory, the number still remains in the accumulator.

2. The same is true when you transfer a number from the accumulator to the B register.

3. When you *read* a number from the memory into the accumulator, it remains in the memory.

4. If there is already a number in a memory location when you *write* into it, the old one is automatically erased first.

5. The same is true when you *read* from the memory into the accumulator. If there is already a number in the accumulator, it is erased.

KEYBOARD ENTRY

When a particular number is going to be used only once in solving a problem, you may want to enter it into the accumulator manually while the problem is being run instead of storing it in the memory and later reading it out. This is what we did with t in the example we just programmed.

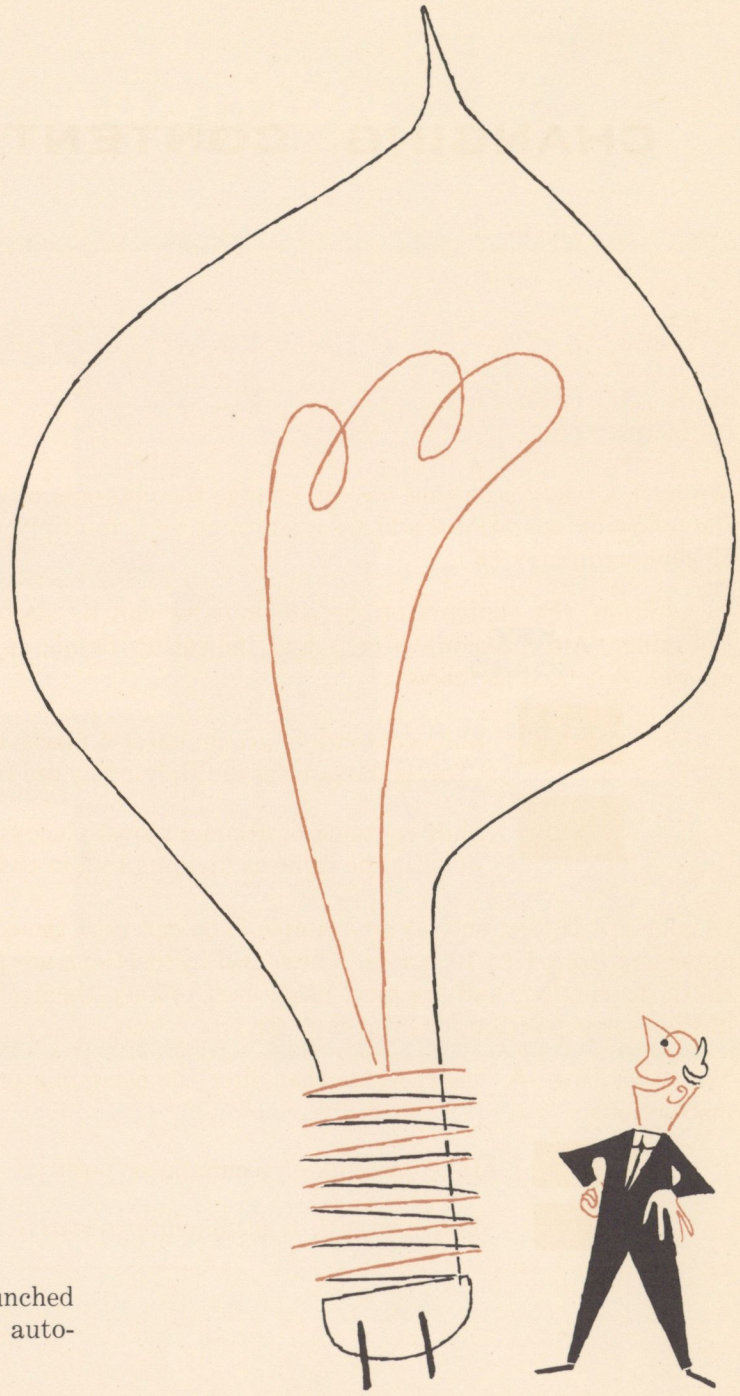
Just as we did with t, we show a **K** in the program to indicate a keyboard entry. When the E101 reaches that instruction, it will stop and the KEYBOARD light will flash, notifying the operator that the machine is ready for a keyboard entry. The operator will then enter the number in the keyboard and touch the motor bar. This not only puts the number into the accumulator, but also prints it on the report.

Actually the operator has a choice of four motor bars. Each one moves the carriage in a different way so as to control the place on the report where the next number is to be printed. The operator selects the one that puts the number in its proper place.

K is also used when loading the memory. If you want to store S_0 in memory location 98 and 1/2 g in 99, you would program the loading as follows:

K	Enter S_0 into the accumulator when the operator puts it on the keyboard and touches the motor bar.
W 9 8	Store S_0 in memory location 98.
K	Enter 1/2 g.
W 9 9	Store in 99.

Note: Data may also be entered into the accumulator by means of punched tape. The optional Tape Input Unit reads and decodes the tape and automatically enters it into the E101.



CHANGING CONTENTS OF ACCUMULATOR

We have already seen that we can change the contents of the accumulator by any of the following operations: add (+), subtract (−), multiply (×), divide (÷), read (**R**) or keyboard entry (**K**).

In addition, the contents of the accumulator can be altered by programming an **A** (meaning "Alter Accumulator") and indicating beside it how contents should be altered. **A** is used as follows:

A 14 — Shift contents of accumulator 4 places to the *left*.
(This is the same as multiplying by ten to the 4th power).

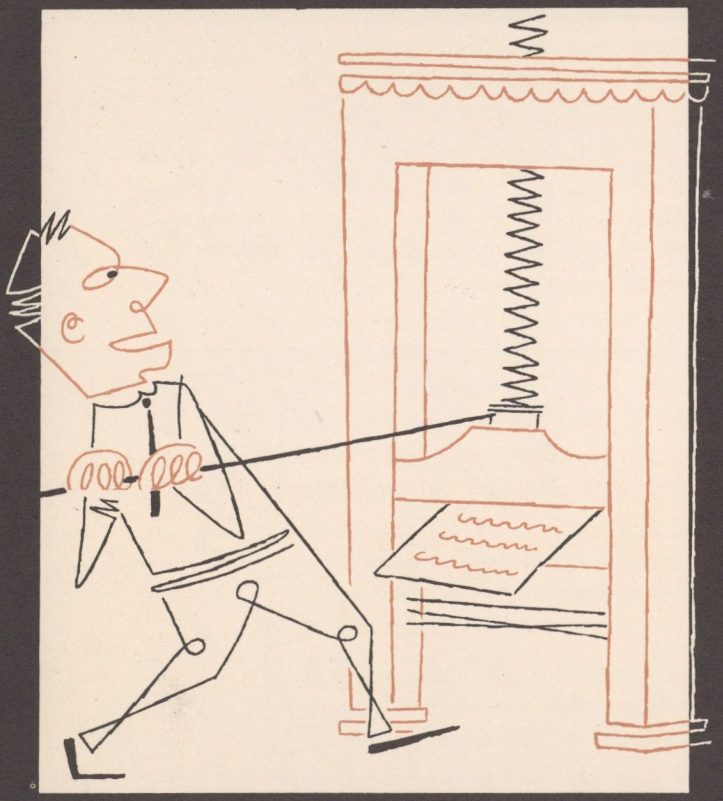
A 24 — Shift contents of accumulator 4 places to the *right*.
(This is the same as dividing by ten to the 4th power).

Of course, 4 is used only as an example. You can shift the contents of the accumulator anywhere from 1 to 10 places. These two instructions are sometimes called "decimal shift" instructions and are most often used to keep the decimal point of your answers in the proper position for print-out, etc.

The other two **A** instructions that alter the accumulator are the "absolute value" instructions:

A 3 — Make contents of accumulator positive.

A 4 — Make contents of accumulator negative.



PRINTING

Whenever you want the E101 to print what is in the accumulator, you program in a **P**. Next to the **P** you indicate which motor bar you want to do the printing. Just as in the case of a keyboard entry (see page 10), the choice of motor bar determines where on the report the machine will print. The four printing instructions are:

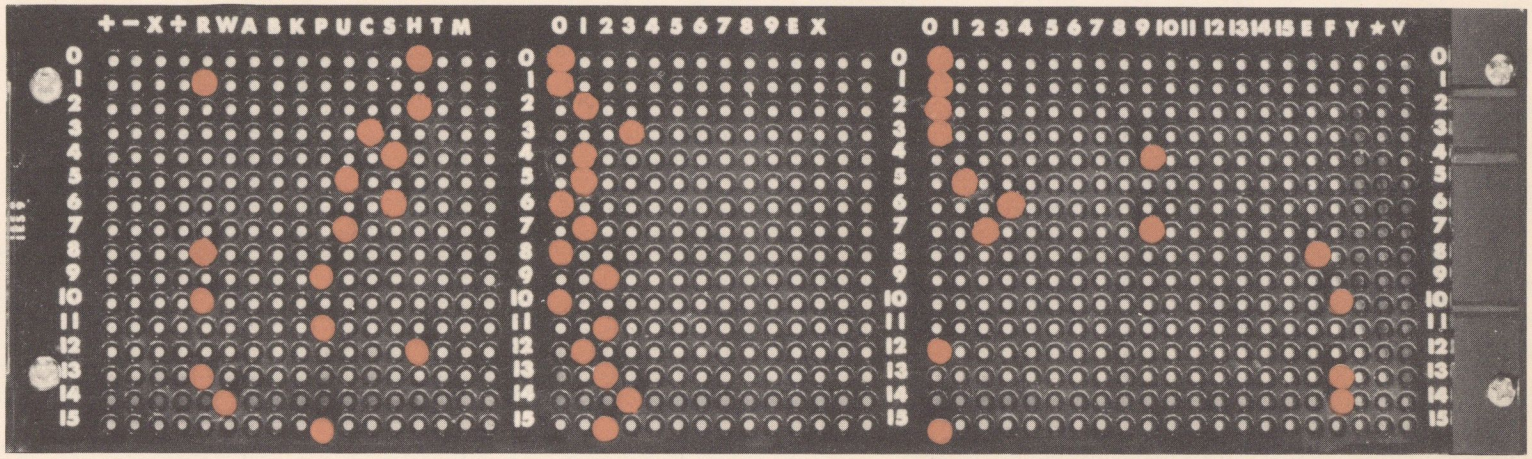
P1 **P2** **P3** **P4**

In addition to printing, each of the above printing instructions positions the carriage to a different place on the form. **P**3 tells the machine to print numbers in a vertical column, **P**2 is used to print across the report horizontally, etc. In programming, you select the printing instructions that will give you a finished report in whatever form you desire. The report itself can be a tape listing, ledger card, duplicating master, continuous snap-out form or practically any other type or size document you wish to use. The E101 prints directly on the document, without necessity of recopying.

Note: In addition to the pinboards, program instructions may be stored on punched tape and read into the E101 by means of the optional Tape Input Unit.

PINBOARDS

Up to this point we have been talking about instructions to the E101, but we have not yet discussed *how* these instructions are given. Actually, this is one of the most important features of the E101. We use pinboards which are light-weight frames (similar to cribbage boards) containing 16 horizontal rows of tiny holes. Each row corresponds to one instruction, making a total of 16 instructions on a pinboard. For each instruction we drop a pin in the hole that corresponds to the operation we want the machine to perform. In that same row we drop a pin in the hole that represents the first digit of the memory location and one for the second digit. For example, if you wanted to tell the machine to subtract the contents of memory location 98, you would drop three pins into the row you are using for that instruction: one for the minus sign, one for the 9 and one for the 8.



Once you have placed the pins in the pinboards, the next step is to insert the pinboards in the E101 (there is room for eight pinboards in all). The machine then follows each instruction in sequence *automatically*, unless told to do otherwise.

When programming, we generally write down our instructions (just as we've been doing) on an E101 program sheet, transfer the information to a prepunched template which fits on the pinboard, drop the pins in place, and insert the assembled pinboards in the machine.

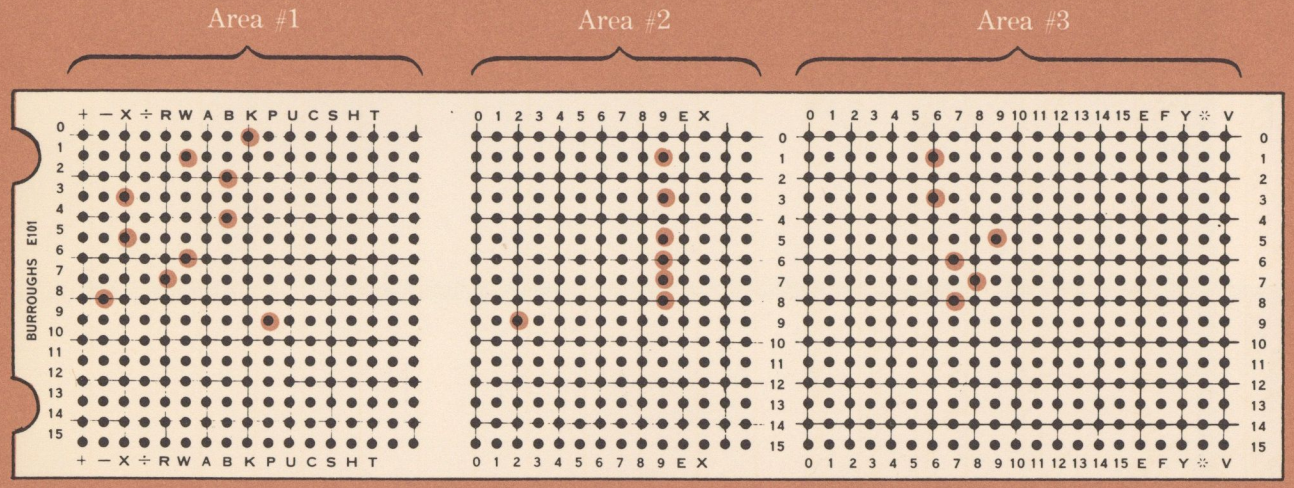
PINBOARD 1

E101 PROGRAM SHEET

Step	1	2	3	Explanation
0	K			Enter t into the accumulator when operator enters it into the keyboard and touches the motor b
1	W	9	6	Write t in 96
2	B			Transfer t from accumulator to B register.
3	X	9	6	Multiply t in B register by t in 96
4	B			Transfer t^2 from accumulator to B register
5	X	9	9	Multiply t^2 in B register by $\frac{1}{2}g$ stored in 99
6	W	9	7	Write the answer, $\frac{1}{2}gt^2$, in 97
7	R	9	8	Read S_0 stored in 98 into accumulator
8	−	9	7	Subtract $\frac{1}{2}gt^2$ in 97 from S_0 in accumulator
9	P	2		Print answer, using motor bar 2.
10				
11				
12				
13				
14				
15				

E101 PROGRAM SHEET

TEMPLATE



As shown above, each horizontal line represents one instruction. The pin for the operation itself (**+**, **−**, **W**, **R**, **B**, etc.) is always placed in area #1. The pin for the first digit of the memory location is placed in area #2 on the same line. The pin for the second digit of the memory location is placed in area #3.

After a problem has been run, the pinboards may be removed from the machine and stored. Or, if the problem is not going to be run again anytime soon, pins may be removed and only the templates filed for later use. Meanwhile, pinboards and pins can be re-used for other problems.

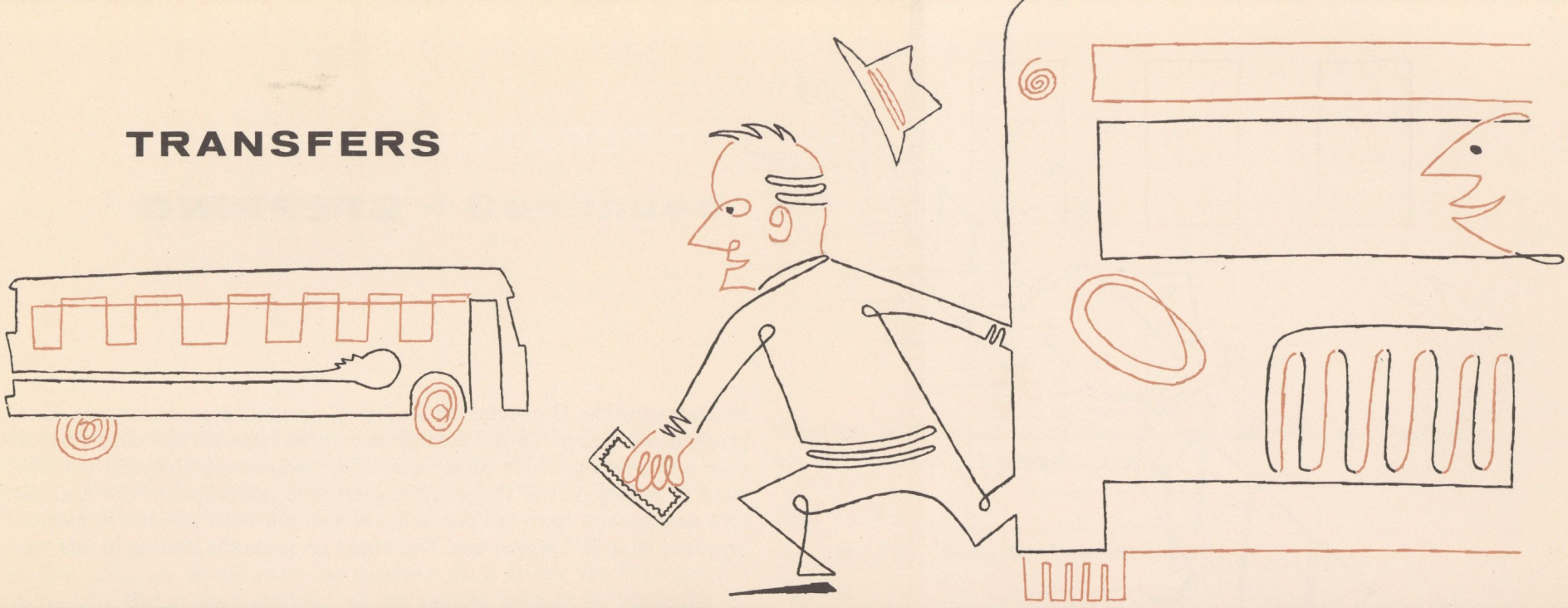
OPERATOR OVERRIDE

One of the outstanding features of the Burroughs E101 is operator override. Even though the machine computes and prints completely automatically, you can program in an instruction that will halt the machine at a strategic point in the problem. The operator can then use his *judgment* to alter the course of the solution of the problem. When he is ready for the E101 to continue with the problem, he merely touches one of nine START buttons (depending on what he wants the E101 to do next).

The halt instruction is written as:

A *

TRANSFERS



As mentioned earlier, the E101 follows each instruction on the pinboards in sequence *automatically* unless told to do something else.

There are two transfer instructions which tell the machine to go somewhere else instead of to the next instruction. The first of these two instructions is an Unconditional transfer and is written as **U**. If you are in pinboard 2, step 8, for example, and wish to go back to pinboard 1, step 6, you program step 8 of pinboard 2 as: **U** 1 6.

The **U** instruction generally provides the method of going from the last step of one pinboard to the first step of the next pinboard. For example, *P. B. 1*, step 15, will probably read: **U** 2 0.

Another place where **U** is frequently used is to repeat part of a program over and over again. Let's go back to our original problem where we want to compute S at different intervals of time. We have already programmed the problem for one value of t. If we want to program it so that we can compute S for a series of values of t, all we have to do is add one more step to our program—a **U** instruction telling the E101 to go back to the beginning of the problem for another value of t.

Our program would be as follows:

Step	*P. B. 1*	
0	**K**	Enter t in **K**eyboard.
1	**W** 9 6	
2	**B**	
3	× 9 6	Multiply t in 96 by t in B register.
4	**B**	
5	× 9 9	Multiply 1/2 g in 99 by t^2 in B register.
6	**W** 9 7	
7	**R** 9 8	**R**ead S_0.
8	− 9 7	Subtract 1/2 gt^2 from S_0.
9	**P** 2	
10	**U** 1 0	Go back to step 0 for next value of t.

STEPPING

In discussing the **U** instruction we found we could repeat any part of the program we desired. In the example shown we repeated the same program over and over again for each value of t. Sometimes, however, you may want to *modify* part of the program each time you repeat it. For example, let's assume you have ten constants which you want to load in memory locations 90 to 99 for later use. You might program the loading in this way:

Step	*P. B. 1*
0	**K**
1	**W** 9 0
2	**K**
3	**W** 9 1
4	**K**
5	**W** 9 2
6	**K**
7	**W** 9 3

Step	*P. B. 1*
8	**K**
9	**W** 9 4
10	**K**
11	**W** 9 5
12	**K**
13	**W** 9 6
14	**K**
15	**U** 2 0

Step	*P. B. 2*
0	**W** 9 7
1	**K**
2	**W** 9 8
3	**K**
4	**W** 9 9

You can easily see that you are actually repeating the same two steps of your program (**K** and **W**) over and over again. In each case, however, you are *modifying* the **W** instruction slightly by writing first into 90, then

STEPPING – Continued

into 91, etc. In order to make use of our **U** instruction and thereby save program space, we can use an *automatic* internal switch called the F switch. This switch makes it possible to *change* the second digit of the memory location each time the routine is repeated.

The first thing we would have the machine do is to set the F switch at zero, or " **H**ome the F switch," as we usually refer to it. We write this instruction:

H 1 0 **H**ome the F switch to zero.

In order to change the second digit of the memory location each time we repeat the routine, we want to advance the value of the F switch once each time. We use the **S** instruction for this purpose. It is written as follows:

S 1 9 **S**tep the F switch to the next value each time the E101 comes to this part of the program. Do this until the F switch steps past 9.

Using the **H**, **S** and **U** instructions, we can now write our program in five steps instead of twenty-one steps:

Step	*P. B. 1*		
0	**H**	1 0	**H**ome the F switch to zero.
1	**K**		
2	**W**	9 F	Store constant in memory location 9F where F is whatever position the F switch is in. The first time it would be 0, then 1, etc.
3	**S**	1 9	**S**tep the F switch once each time this instruction is executed until the F switch steps past 9.
4	**U**	1 1	Go back to instruction 1 and repeat routine.

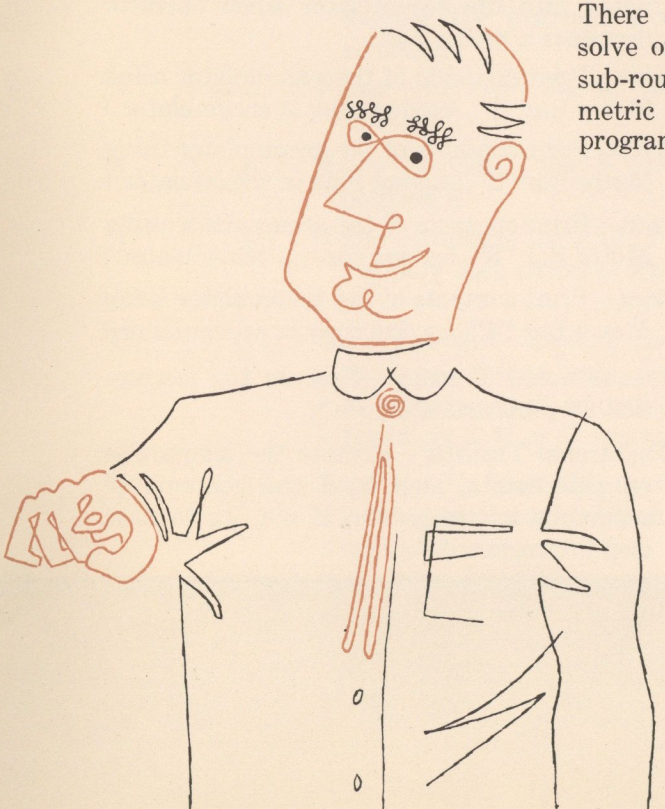
In addition to the F switch, there is an E switch which we generally use to change the first digit of the memory location instead of the second. When a problem requires more than ten memory locations, both switches are frequently used.

CONDITIONAL TRANSFER

We spoke earlier of two transfer instructions. The first is **U** which we have discussed. The other is a Conditional transfer we refer to as **C**. Like **U**, it tells the E101 to go to another instruction. However, it tells the machine to go there *only* on one condition—if the number in the accumulator is negative.

This instruction is very useful in specialized cases. Let's suppose we are preparing the weekly payroll for a large company using the E101. After we arrive at each employee's weekly salary and his accumulated salary to date, we have to determine whether or not to deduct social security tax. If his accumulated salary to date is less than $4200, we want to deduct the tax. If, however, it has already reached $4200, we do not want to make the deduction. The E101 will make the decision automatically if we use **C** in our programming. We would store 4200 as a constant value in some memory location and program the machine to subtract 4200 from the employee's accumulated salary. The next step in the program would be **C**. As long as the difference between the accumulated yearly salary and 4200 is negative (meaning total salary is less than $4200), the E101 will transfer to that part of the program that computes social security tax. But if this result is positive (indicating that salary is over $4200), the E101 will skip over the social security deduction and go on with the rest of the program.

CONCLUSION



We have now covered all the essential characteristics of programming the Burroughs E101. As you can see, you don't need an advanced degree in mathematics or electronics. With a little practice, YOU can now program the E101!

There are many problems you will want to solve on the E101 that have as part of them sub-routines such as square roots, trigonometric functions, etc. There is no need for programming these sub-routines each time they occur. Instead, you would program them once and then use the appropriate programs whenever necessary. There is a library of these subroutines which has already been programmed and is available to anyone interested in using the E101.

Also available is additional programming material on decimal point scaling and the tape input unit. For further information concerning the E101, write to:

ElectroData Division
Burroughs Corporation
1616 Walnut Street
Philadelphia 3, Pennsylvania

Note: If you want to practice programming problems of your own, a complete list of the E101 instructions follows on the next two pages.

SUMMARY OF E101 PROGRAM INSTRUCTIONS

INSTRUCTION			OPERATION
+	a	b	Add—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—Subtract contents of ab from contents of accumulator.
×	a	b	Multiply—Multiply the contents of the B register by the contents of memory location ab (answer is in the accumulator).
÷	a	b	Divide—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	**R**ead—Read the contents of memory location ab into the accumulator (leaving copy in ab).
W	a	b	**W**rite—Write the contents of the accumulator into memory location ab, leaving copy in accumulator.
A	1	b	Shift Left—Shift the contents of the accumulator b places to the left.
A	2	b	Shift Right—Shift the contents of the accumulator b places to the right.
A	3		**A**bsolute Value—Make the contents of the accumulator positive.
A	4		Negative of **A**bsolute Value—Make the contents of the accumulator negative.
A		*	Halt—Halt the machine.
B			**B** Register — Transfer the contents of the accumulator into the B register, leaving copy in accumulator.
K			**K**eyboard—Transfer the contents of the Keyboard into the accumulator when operator depresses a motor bar.[1]
P	1		**P**rint—Print contents of the accumulator using Motor Bar "1", leaving copy in accumulator.[2]
P	2		**P**rint—Print contents of the accumulator using Motor Bar "2", leaving copy in accumulator.[2]
P	3		**P**rint—Print contents of the accumulator using Motor Bar "3", leaving copy in accumulator.[2]
P	4		**P**rint—Print contents of the accumulator using Motor Bar "4", leaving copy in accumulator.[2]
U	a	b	**U**nconditional Transfer—Execute the instruction on pinboard a, step b.[3]
C	a	b	**C**onditional Transfer—Execute the instruction on pinboard a, step b, if the contents of accumulator is negative; if not, go to next step in program.[3]

SUMMARY OF E101 PROGRAM INSTRUCTIONS.....Continued

INSTRUCTION			OPERATION
S	0	b	**S**tep—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	**S**tep—Same as above, but using the **F** switch.
S	2	b	**S**tep—Step **E** and **F** switches once; if E≠b+1, execute the next instruction; if E=b+1, execute the instruction after the next instruction.
H	0	b	**H**ome—Advance the **E** switch, stopping at position b.
H	1	b	**H**ome—Advance the **F** switch, stopping at position b.
H	2	b	**H**ome—Advance the **E** and **F** switches together, stopping both switches when **E** is at position b.
T		11	**T**ape 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 the tape.
T		12	**T**ape Read—Read the next number on tape into accumulator and then continue with the next pinboard instruction.

NOTES:

[1]Pinning 0 in area 3 affects a non-print for this operation.

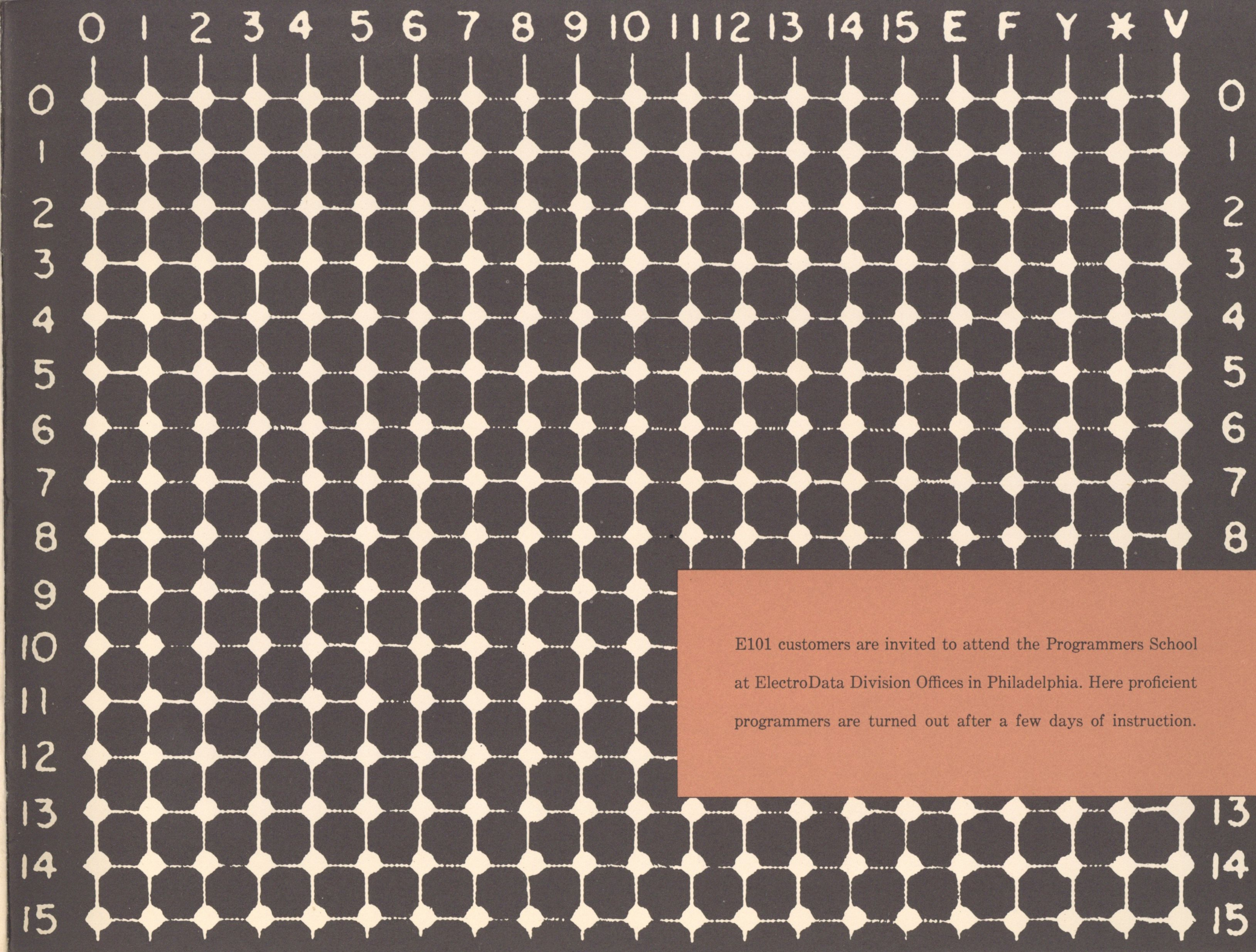
[2]Pinning 0 in area 3 gives a non-print, while the * gives a print-and-halt.

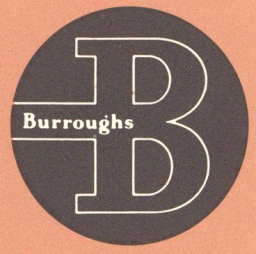
[3]Pinning 0 in area 2 keeps the transfer (to instruction b) in the same pinboard, while the * in area 3 transfers to whatever instruction was last executed (in pinboard a)+1.

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.

"SEE, YOU CAN PROGRAM THE E101"


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 E F Y ✱ V
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
E101 customers are invited to attend the Programmers School at ElectroData Division Offices in Philadelphia. Here proficient programmers are turned out after a few days of instruction.



ElectroData Division
Burroughs Corporation
1616 Walnut Street
Philadelphia 3, Pennsylvania

Form No. EDP 104

6-9-5

Litho in U.S.A.