445

PRELIMINARY OPERATING

MANUAL

OPERATING MANUAL

FOR THE

COMPUCORP 445 STATISTICIAN

TABLE OF CONTENTS

INTRODUCTION

Congratulations. You've just bought an extremely versatile piece of calculating equipment. Now all you have to do is learn how to use it. But -- as you're about to find out -- using the Compucorp 445 Statistician isn't much more difficult than using your pencil. (In some cases, it's easier.) And it's certainly a whole lot faster.

The reason it's so easy to use is simply because it was designed from the ground up to do things your way -- rather than the other way around. So if you understand a particular problem, you can probably do it on the 445 without much fooling around with mechanical intricacies. Just enter the information, tell the machine what to do with it and read the answer. Of course, there's a right way to go about doing this. And there are a few little tricks that save a lot of time and eliminate several steps on certain operations. So even though the 445 is set up to follow your normal sequences, it's still a good idea to read this manual first so you can get the most out of your machine.

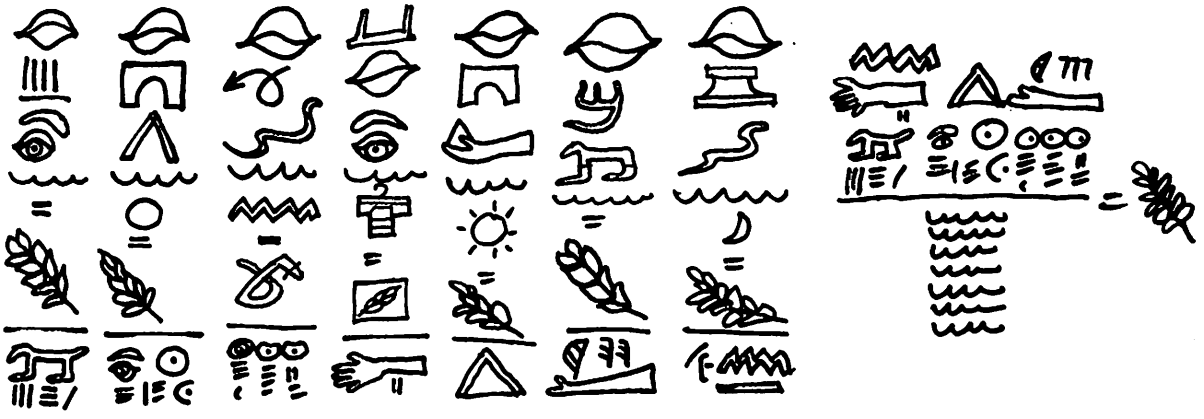
And getting the most out of your machine means more than just getting fast answers. Because once you've mastered the technique of communicating with the 445 you'll discover another interesting aspect of our little machine. Besides being a lavishly powerful little bag of wires, it's a heck of a lot of fun!

CHAPTER ONE

THE GRANARIES OF ISIS

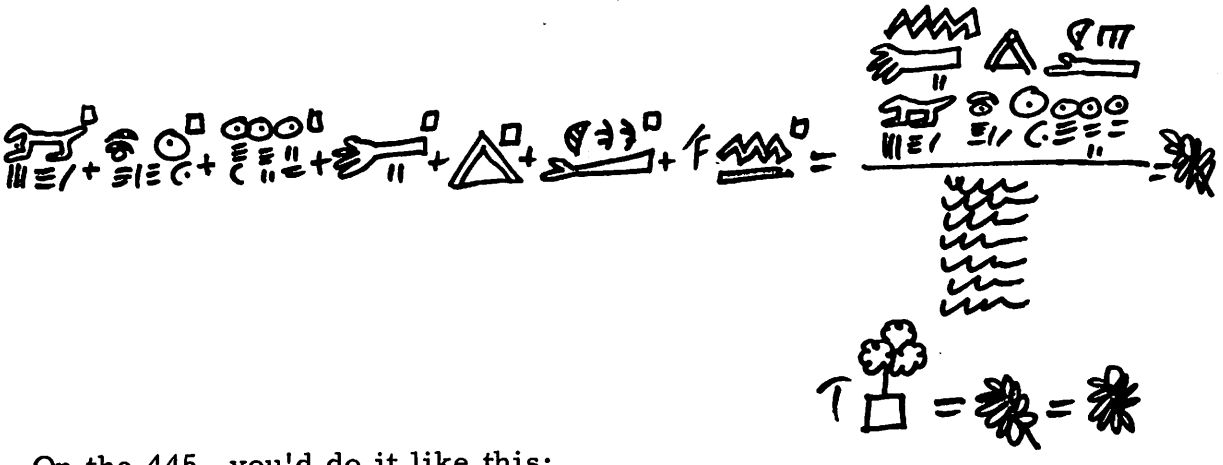
Back in the ancient land of Egypt, the Pharaohs had great granaries built to store surplus wheat during the good years so that they might feed their people during the lean years. One day, the Pharaoh called upon the Keeper of the Granaries and asked him how things were going. He also wanted to know how much wheat was produced during the year, and how much would have to be stored. And he asked him a new question. One that had never occurred to a Pharaoh to ask before. "How much wheat do we usually get, and can we tell beforehand how much wheat we'll have to store so we can build enough grain houses in time to hold it? "

To the first question, the Keeper of the Granaries was able to give a relatively fast answer. He happened to have with him records of grain production for the previous seven years, so he simply took a mathematical average. He did it like this:



While the Keeper of the Granaries was occupied with these calculations, another idea occurred to the Pharaoh. "I notice, O Keeper of the

Granaries, that there is often quite some difference between one year and the next. Is there not some way we can tell how much faith to put in our answer?" The Keeper of the Granaries looked at the Pharaoh with injured eyes, but nonetheless promised to give the Pharaoh an objective measure of the accuracy of his answer. Accordingly, the Keeper of the Granaries then calculated the standard deviation. He did it like this:



On the 445, you'd do it like this:

YEAR (now = 0)	AMOUNT OF GRAIN
year -1	962,300 hekats
year -2	1,127,000 hekats
year -3	627,000 hekats
year -4	805,300 hekats
year -5	719,700 hekats
year -6	1,072,500 hekats
year -7	984,300 hekats

SET GROUP 1 tells the machine where to store the information.

Φ_n 0 clears the areas where information is to be stored.

9 6 2 3 0 0 Σ_{nxx^2}

1 1 2 7 0 0 0 Σ_{nxx^2}

6 2 7 0 0 0 Σ_{nxx^2}

8 0 5 3 0 0 Σ_{nxx^2}

7 1 9 7 0 0 Σ_{nxx^2}

1 0 7 2 5 0 0 Σ_{nxx^2}

9 8 4 3 0 0 Σ_{nxx^2}

the sigma key enters summation data

SD MEAN does what it says (prints SD)

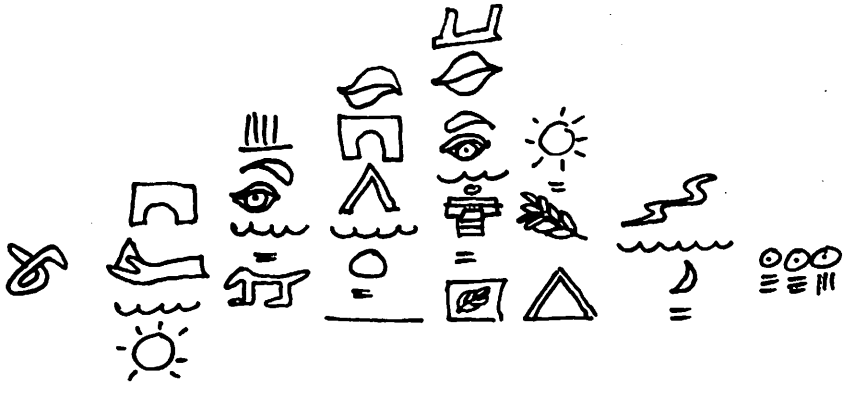
2ND FUNC prints mean

0.00		CL
962,300.00	Σ	1
1,127,000.00	Σ	1
627,000.00	Σ	1
805,300.00	Σ	1
719,700.00	Σ	1
1,072,500.00	Σ	1
984,300.00	Σ	1
186,314.11	SDn-1	1
899,728.57	F	2

On the basis of this information, and considering that the populace consumed around 700,000 hekats of grain each year, the Pharaoh then ordered that 200,000 hekats of grain be stored each year to accommodate the general yearly need.

When he got home, the Keeper of the Granaries began thinking of all

the grain the Pharaoh expected him to store. And he decided that there had to be a better way to calculate the amount of grain needed in any particular year. "There must be a way," he thought, "to know how much grain will be produced each year." So he sat down and studied the records that he had been keeping during all the twenty-three years he had been the Keeper of the Granaries. And he also studied the records his father had kept before him. And he discovered an interesting thing. Every year that the Nile rose very high, there was a lot of grain. And every year that the Nile didn't rise high, there wasn't much grain. The Keeper of the Granaries now had enough information to give the Pharaoh (and himself) a far better way of keeping track of the grain they would need. He wrote down the height of the Nile during each of the previous seven years, and next to each of those numbers he wrote down the amount of grain produced. Then he correlated those numbers. He did it like this:



On the 445, you'd do it like this:

HEIGHT OF NILE	HEKATS OF GRAIN
4.8 cubits	962,300
5.5 cubits	1,127,000
3.6 cubits	627,000
4.0 cubits	805,300
3.5 cubits	719,700
5.9 cubits	1,072,500
4.6 cubits	984,300

[Σn] [0] clears areas where data will be stored.

[ADV] separates CL from data (on tape)

[4] [.] [8] [XY]

[9] [6] [2] [3] [0] [0] [=]

[5] [.] [5] [XY]

[1] [1] [2] [7] [0] [0] [0] [=]

[3] [.] [6] [XY]

[6] [2] [7] [0] [0] [0] [=]

The XY key enters dependent data. The independent variable goes in with the XY key; then the dependent variable goes in with the equals key. The tape skips a space after each pair.

0.00	CL
4.80	X
962,300.00	Y
5.50	X
1,127,000.00	Y
3.60	X
627,000.00	Y

4 XY	4.00	X
8 0 5 3 0 0 =	805,300.00	Y
3 . 5 XY	3.50	X
7 1 9 7 0 0 =	719,700.00	Y
5 . 9 XY	5.90	X
1 0 7 2 5 0 0 =	1,072,500.00	Y
4 . 6 XY	4.60	X
9 8 4 3 0 0 =	984,300.00	Y

dependent data (continued)

SET GROUP 1 tells the machine where first group of data is

LIN REG takes correlation coefficient between X and Y — 0.94 LR 1

2ND FUNC gives us the slope — 190,121.91 F 2

Φ_n 3 gives us the intercept of X on Y — 33,315.83 F 3

Therefore, the equation for estimating the amount of grain that could be expected each year would be

$$\text{Hekats of Grain} = 190,219.91 \times (\text{ht. of Nile}) + 33,315.83$$

With that information, he went to see the Pharaoh once again. "O great Pharaoh," he said, "what dummies we were. There is a great deal more to knowing how much grain to store than we thought. Let me tell you what I have discovered." And the Keeper of the Granaries then explained to the Pharaoh how whenever the Nile floods a lot they get a lot of grain, and whenever it floods just a little, they get just a little grain. And he showed him the table of numbers which would predict the amount of grain that will be produced for each height the Nile might reach.

"O Keeper of the Granaries," the Pharaoh said, after studying the tables for a very long time, "what a dummy you are. Nowhere in your great table of numbers do you account for the Omens of Isis. How are we to value your table of numbers if you leave out Isis?" The Keeper of the Granaries, knowing of the great power of the Priest of Isis, promised that he would undertake to consider the Omens of Isis and bring the Pharaoh a new table by which to judge the production of grain. So, grumbling somewhat under his breath, the Keeper of the Granaries gathered up his table and went back to his home. And he sent for the Priest of Isis.

The Keeper obtained from the Priest of Isis a record of the Omens of Isis for the past seven years, and sent him away. Then he tried to correlate these with the data he had already collected. He did it like this:

On the 445, you do it like this:

OMENS*	HEIGHT OF NILE (cubits)	GRAIN (hekats)
+4	4.8	962,300
+2	5.5	1,127,000
+7	3.6	627,000
-3	4.0	805,300
+1	3.5	719,700
+4	5.9	1,072,500
+3	4.6	984,300

(*Good agricultural omens minus bad agricultural omens for the six month period preceding the harvesting of the crop.)

[Σn] [0] clears areas where data will be stored.

[ADV] separates CL from data

[4] [XY]

[4] [.] [8] [XY]

[9] [6] [2] [3] [0] [0] [=]

[2] [XY]

[5] [.] [5] [XY]

[1] [1] [2] [7] [0] [0] [0] [=]

[7] [XY]

[3] [.] [6] [XY]

[6] [2] [7] [0] [0] [0] [=]

Three-variable dependent data are entered the same way as two-variable, using the XY key for the independent variables and the equals key for the dependent variable. As before, the tape skips a space after each group of data.

0.00	CL
4.00	X
4.80	Y
962,300.00	Z
2.00	X
5.50	Y
1,127,000.00	Z
7.00	X
3.60	Y
627,000.00	Z

CHG SIGN * 3 XY		-3.00	X
4 XY		4.00	Y
8 0 5 3 0 0 =		805,300.00	Z
1 XY		1.00	X
3 . 5 XY		3.50	Y
7 1 9 7 0 0 =	dependent data (continued)	719,700.00	Z
4 XY		4.00	X
5 . 9 XY		5.90	Y
1 0 7 2 5 0 0 =		1,072,500.00	Z
3 XY		3.00	X
4 . 6 0 XY		4.60	Y
9 8 4 3 0 0 =		984,300.00	Z

(*Since entry on the 445 is algebraic, the minus sign indicates subtraction rather than a negative value. To make a number negative, you have to use CHANGE SIGN.)

SET GROUP	2	LIN REG	correlates Nile and Grain, as before.
SET GROUP	3	LIN REG	correlates Omens and Grain.

0.94	LR	2
-0.04	LR	3

Noticing absolutely no correlation whatever between the Omens of Isis and the production of grain, the Keeper simply drew a symbol of the benevolence of Isis on his table and presented it again to the Pharaoh, saying, "O great Pharaoh, I have considered the Omens of Isis, and the table before you does not in any way contradict their portent." The Pharaoh was very pleased, and ordered that just such an amount of grain as was predicted by the Keeper's table be stored each year according to the height to which the Nile rose, and with the grace of Isis.

After three years, when the Pharaoh noticed that the Keeper's table was, indeed, marvellously accurate, he sent for this Keeper of the Granaries and made him Wizard of the Nile. And so he remained until the end of his days.

If the Keeper of the Granaries had had a 445 (and the same data was in his machine as is presently in ours) he could have predicted the grain production for each year like this:

[SET GROUP] [4] indicates three-variable data		
[4] [Σn] [9] first independent variable	4.00	X
[5] [.] [1] [Σn] [9] second independent variable	5.10	Y
dependent variable automatically supplied because of [Σn] [9] above.	990,630.65	Z 4

With a machine like this, who knows, maybe he would have become the new Pharaoh.

CHAPTER TWO

MORE ABOUT KEYS AND THINGS

As you may suspect, there's much more to the 445 than you can immediately see on the keyboard. In addition to all the normal arithmetic operations, there's a whole bunch of statistical operations built right in. You can do simultaneous summations for three distinct groups of data. You can enter data into these three groups either singly or in "grouped" form, or as two- or three-variable dependent data. And you can delete data items from each of the three groups. Using the data in each group, you can calculate the mean, standard deviation (either n or n-1 method), standard error of the mean, z-statistic and normal probability associated with the z-statistic. Also, you can calculate independent and dependent t-statistics between any two of the groups. And you can do either two-variable or three-variable linear regression with the ability to "extend" the regression to calculate dependent variables from the independents. And in addition to all this, you can do chi-square goodness-of-fit and permutation/combination calculations.

Besides the capacities of the machine, there are many ways in which operation of the machine can be made much more sophisticated than the techniques we've been using. So before we get into programming -- which in itself is a very powerful story -- let's go over the keyboard in an organized kind of way and consider all the little nooks and crannies in our 445's storehouse of tricks.

SETTING IT UP

Assuming you've plugged the machine into a power source, the first thing you have to face is the ON-STANDBY-OFF switch at the very left of the top row of buttons. In the ON position, the 445 will be completely operative. All keys will operate, and all programming functions will function. On STANDBY, no keys or programming functions will operate, but the machine will retain in memory any program instructions or data that have been fed into it while it was ON. When you switch the machine from STANDBY to ON, RESET is automatically executed -- which clears all signals which may have been lurking in any of the operation circuits or in the entry register. Then you can just continue using whatever data or program instructions you left in the machine before putting it on STANDBY.

When you put the machine on OFF, all power is cut off and whatever information may have been stored in any of the memory circuits is completely lost. When you switch the machine from OFF to ON, several significant things happen. All registers are cleared (we'll talk about registers a little later). All program memory is filled with NOOP codes. This means that there is nothing -- absolutely nothing -- in program memory. (You'll find out why this is important when we get to programming.) The decimal point is set to two places. Therefore, all numbers will be printed (when they're printed) with two digits to the right of the decimal point. Group 1 is set (more about Set Groups later). And RESET is executed.

(Besides clearing the operation circuits and the entry register, RESET will also clear an ERROR or OVERFLOW condition. If you press [RESET] in the middle of a multiple-key sequence, the machine will abandon the operation in progress and clear it all out so you can start over from the beginning. We'll talk more about ERROR and OVERFLOW later on.)

Now we have to consider the rest of that top row of switches. For keyboard operation you can ignore most of them. But there are three that have to be checked first. The fourth one from the very right-hand side, the RUN-STEP-LOAD switch, should always be on RUN when you're doing keyboard operations which do not involve programming. And if your machine has a DEGREE/GRAD switch, you have to decide which setting you want. With this switch in the DEGREE position, all trigonometric functions are calculated on a base of the 360° circle. With the switch in the GRAD position, these trigonometric functions are calculated on a base of the 400-grad circle -- with 100 minutes per grad, and 100 seconds per minute. Also, there are several choices to make and things to know about printing.

PRINTING

The 445 printer will print the mantissa of numbers with up to ten digits of significance, the two-digit exponent (when required), minus sign for both exponent and mantissa (when negative) and print symbols indicating the operation performed.

Print Format

When you plugged the machine in, the decimal point was set with two places to its right. From then on, you're on your own. To establish the position of the

decimal point, all you have to do is push [SET D. P.], followed by a single digit (0-9). Digits 0-8 indicate the number of digits to the right of the decimal point. Digit 9 tells the machine to print everything exponentially.

Regardless of how you set the decimal point, however, the 445 will do a few convenient things all by itself. Fractional numbers are always printed with at least one digit of significance. If necessary, the decimal point will be shifted to the left automatically until this is achieved. If the decimal point has to be shifted beyond the tenth position in the mantissa to find a digit of significance, then printing will automatically shift to exponential. In exponential mode, the 445 will print the ten most significant mantissa digits, the two exponent digits and a sign for both mantissa and exponent (if negative). Also, if the number is too large to fit to the left of the decimal point as it is set, then the point will be shifted to the right until there are enough places to print the number. If the decimal point must be moved beyond the right-hand margin, then printing will automatically shift to exponential.

Commas will be printed where they belong to the left of the decimal point. And the minus sign of negative numbers will appear next to the left-most digit.

Print Control

With the RUN-STEP-LOAD switch in RUN position, printing is controlled

primarily by the position of the PRINT switch. It can be "ON" or "OFF". With the PRINT switch ON, most keyboard operations will print, providing an audit trail of what you did and how you did it. There are, however, a few keyboard operations which will not print even with the PRINT switch ON. These are:

-CLEAR ENTRY

-SET D.P.

-FLAG*

-HALT*

-BRANCH*

-JUMP*

-RESUME*

With the PRINT switch OFF, no keyboard operations will print except a few functions which print in all cases. These "print always" functions are:

-PRINT ENTRY

-PRINT ANSWER

-IDENTIFIER*

-DOT PRINT*

-OVERFLOW

-ERROR

---

*These are programming operations, and will be covered in a later chapter.

PRINT ENTRY is simply a key that tells the machine to print whatever is in the entry register. It can be used to record selected entries with the PRINT switch OFF. Or, if you haven't entered a new number before pushing [PRINT ENTRY], then the 445 will print the last number it has seen -- either the last number entered or the result of the last operation performed. [PRINT ANS] does the same thing, except that whereas PRINT ENTRY will print the value with the number of places indicated by the decimal point setting without rounding off the final digit, PRINT ANSWER will print the value rounded off to the number of places indicated by the decimal setting. And [PRINT ANS] will leave the rounded number in the entry register, whereas [PRINT ENTRY] will leave the full, unaltered original value in the entry register even though it may have caused a shorter version to be printed.

CLEAR ENTRY is a key that you use when you make a mistake while entering a number. Let's say you want to enter 345.67, and by mistake you hit 345.77. If you catch the mistake before it's printed, just push [CLEAR ENTRY]. The entry register will be cleared, and you can enter the number all over again. CLEAR ENTRY clears only the number being entered, it will not affect an operation in progress. CLEAR ENTRY can also be used to clear an ERROR or OVERFLOW condition.

ERROR condition results from an attempt to engage the 445 in an illegal operation. When that happens, "ERROR" will be printed, regardless of

the PRINT switch position. The calculator operation will halt. The keyboard will become inoperative. And the IDLE light will flash. (The IDLE light is that rectangular little light on the upper right-hand side.) Either RESET or CLEAR ENTRY will clear the ERROR condition. These are the illegal operations which cause ERROR:

- Divide by zero
- 1/x of zero
- $\sqrt{\ }$ of a negative number
- Log of zero or negative number
- Entry of more than thirteen digits (or a decimal point followed by more than twelve digits)
- Calculating 0^{-x} with the $\boxed{a^x}$ key
- Calculating 0^0 with the $\boxed{a^x}$ key
- Calculating standard deviation (n-1) with $n \leq 1$
- Calculating standard deviation (n) with $n \leq 0$
- Factorial of numbers less than zero
- Factorial of non-integer numbers
- Pressing two or more keys simultaneously
- Exceeding the two-level key buffer*
- Calculating linear regression with $n \leq 1$
- Calculating z with $n \leq 1$

---

*See ENTERING NUMBERS, below.

OVERFLOW condition is caused by attempting to operate with numbers outside the range of the calculator. (The range of the 445 is 10^{-98} to 10^{+98}.) When this happens, "OVERFLOW" is printed, regardless of the PRINT switch position. Calculator operation halts. The keyboard becomes inoperative. And the IDLE light flashes. RESET or CLEAR ENTRY will clear the OVERFLOW condition.

(The IDLE light will remain off while the machine is calculating. In most cases, you don't have to pay much attention to it, unless it's flashing. When it flashes, this means the machine has gone into either ERROR or OVERFLOW mode, and you have to clear it before you can do anything else.)

Paper Advance

If you want to advance the paper without printing, just touch the ADVANCE key next to the paper bail. Pressing [ADV] will advance the paper one space. Holding the key down will cause repeat spacing until you let up.

OK. So we've plugged the machine in, turned it on and set PRINT to ON. Now what?

ENTERING NUMBERS

Entering numbers on the 445 is a simple matter of using the 0-9 numeral keys in the middle of the keyboard and the decimal point (plus the

CHANGE SIGN and EXPONENT keys, as explained below). All numbers entered are assumed to be whole numbers unless you hit [.], which you'll find between [0] and [CHG SIGN]. The [.] is a decimal point, which you have to use to enter fractional numbers. CHANGE SIGN, as you might have guessed, changes the sign. It may be pressed any time during entry of the mantissa (also directly before or after it) to make the mantissa negative. If there is a negative number in the entry register and you press [CHG SIGN], it will change this number to positive. If the entry register is positive, [CHG SIGN] will make it negative. The CHANGE SIGN key may also be pressed during entry of an exponent to make the exponent negative.

You can enter a number exponentially by using the EXPONENT key near the bottom of the block of keys just left of the numeral keys. Simply enter the mantissa (with the decimal point in any position), press [EXP], and then enter the exponent (it can be either one or two digits). If you don't enter a mantissa, but just hit [EXP] and enter an exponent, the 445 will assume a mantissa of 1. You can make the exponent negative by pressing [CHG SIGN] after pressing [EXP].

Keyboard Buffering

Most keys are electrically interlocked to prevent multiple, simultaneous depression. That's why the 445 goes into ERROR mode when two or more of them are pressed at once. However, these keys are also

buffered to two levels. This means that during calculation initiated by a key, two more keys may be pressed without loss of operation continuity. As long as no two of the keys are pressed at the same time, the 445 will hold the three keys sequentially and perform the operations they indicate in the order in which the keys were pressed. If you press more than two more keys while the machine is working on something for you already, it'll print "ERROR" and stop everything.

A few keys operate independently of the interlocking and buffering. These are:

-RESET

-CLEAR ENTRY

-PAPER ADVANCE

-LIST PROGRAM

BASIC KEYBOARD ARITHMETIC

Arithmetic operations on the 445 are performed algebraically. You enter values and operations pretty much as you would write them in an equation. For example, suppose you wanted to add 4 and 6. You'd simply press [4] then [+] then [6] and [=]. The 445 will print both of the numbers entered and the result of the operation. The procedure is the same for subtraction, multiplication and division, with the substitution, of course, of [-], [X] or [÷] for the [+]

in our example. Multiplication and division are rounded in the thirteenth digit.

Another key which operates like the basic arithmetic keys is $\boxed{a^x}$, which you'll find just to the right of them. $\boxed{a^x}$ raises a number to a power. You do it like this:

Enter number (a).

Press $\boxed{a^x}$.

Enter power (x).

Press $\boxed{=}$.

That's all there is to it. Both (a) and (x) may be positive or negative, integer or fraction. Except that a negative (a) with a noninteger (x) will cause ERROR. And there are also the following little facts to keep in mind:

$a^0 = 1$

$a^1 = a$

$1^x = 1$

$0^x = 0$

0^{-x} = ERROR

0^0 = ERROR

Just to the right of $\boxed{a^x}$ is $\boxed{\frac{1}{x}}$. This is the INVERT (or RECIPROCAL) key. It simply takes the reciprocal of the number in the entry register, prints it and leaves it in the entry register. All you do is

press $\boxed{\frac{1}{x}}$ and the number in the entry register gets the reciprocal treatment.

To the right of $\boxed{\frac{1}{x}}$ is $\boxed{\sqrt{\ }}$, which -- not surprisingly -- takes the square root of the number in the entry register. All you have to do is press $\boxed{\sqrt{\ }}$ and the number in the entry register will be printed again and then the square root of that number will be printed and left in the entry register.

SCRATCH PAD REGISTERS

In the left-most block of keys, along with PRINT ENTRY, PRINT ANSWER, RESET, etc., there are two keys stacked vertically that get you into the ten scratch pad registers accessible from the keyboard. These keys are $\boxed{STn}$ and $\boxed{RCLn}$. That is, STORE n and RECALL n, where n stands for the name of the register indicated. The scratch pad registers are named 0, 1, 2, 3, etc., up to 9. They are not affected by any keyboard operations except sum-square, XY entry, delete, chi-square and register arithmetic (all of which will come up again later on). To store a number in one of these ten registers, all you have to do is enter the number into the entry register, push $\boxed{STn}$, followed by a single digit to identify the register in which you'd like this number stored. At any later time, you can recall this number from the scratch pad register into the entry register (and therefore onto the tape, if PRINT is ON) simply by pushing $\boxed{RCL_n}$

and the name of the register you put the number into before. It's important to remember that when you use [STn] to put a number into a scratch pad register, the new number will replace anything that was there before. If, for instance, register 6 has 123.456 in it and you enter 32.45, press [STn] and [6], register 6 will now have 32.45 in it. The 123.456 that was there before is completely lost.

EXAMPLE:

[SET D. P.] [3]

[4] [5] [6] [7] [8] [STn] [3]

[RCLn] [3]

[1] [2] [3] [4] [5] [STn] [3]

[RCLn] [3]

```
45678.000   ↓   3
45678.000   ↑   3
12345.000   ↓   3
12345.000   ↑   3
```

Notice that the 445 prints the number of the register you use as well as a symbol indicating the operation performed. ↓ means STORE and ↑ means RECALL.

It's also possible to add to a number stored in a scratch pad register. And you can multiply it, divide it and subtract from it. That's the subject of our next section.

(To be perfectly honest, there aren't really ten scratch pad registers. There are eleven. The decimal point also names a register which has another use we'll find out about later on. But when it's not being used that way, you can use it exactly like any of the other ten scratch pad registers. Just press [.] instead of a numeral key in conjunction with the access keys described above and it's at your service.)

Register Arithmetic

You can perform all the basic arithmetic operations on the values stored in the scratch pad registers by using [ST_n] in conjunction with the specific arithmetic function key. Fox example, suppose you wanted to store 4442 in register 0 and then do things to it. You'd press:

[4] [4] [4] [2] [ST_n] [0]

Now, if you wanted to add 2 to this number, all you have to do is press:

[2] which is the value you're adding

[ST_n] which tells the machine you're going to send it to a scratch pad register

[+] which tells the machine what to do when it gets there

[0] which tells the machine which register to do it in.

You now have 4444 in register 0. Let's divide it in half. Just go through

the same steps as above, substituting [÷] for [+] . (This will cause the number in register 0 to be divided by the number in the entry register.) Now register 0 has 2222 in it. Shall we multiply it times four? OK. Do the same thing again, this time starting off by entering 4 into the entry register, then going through the same sequence, only substituting [X] for [+] . Now press [RCL_n] and [0] . You should have 8888 in the entry register and on the tape. It's important to note that in every case [ST_n] was pressed before the arithmetic function key. And the name of the register after the arithmetic function. This is the only way it will work.

Here's the tape which shows all the arithmetic we just did:

[4] [4] [4] [2] [ST_n] [0]	4442.000		↓ 0
[2] [ST_n] [+] [0]	2.000	+	↓ 0
[RCL_n] [0]	4444.000		↑ 0
[2] [ST_n] [÷] [0]	2.000	÷	↓ 0
[RCL_n] [0]	2222.000		↑ 0
[4] [ST_n] [X] [0]	4.000	X	↓ 0
[RCL_n] [0]	8888.000		↑ 0

We just recalled 8888 from register 0 into the entry register. Do it again. You'll notice that you get 8888 again. Recalling a number from a register doesn't disturb the register.

We can use numbers stored in registers in another way. We just used a number in the entry register to add to, subtract from, divide or multiply a number in a scratch pad register. We can also do the same thing in reverse -- we can use a number in a scratch pad register to add to, subtract from, divide or multiply a number in the entry register. The procedure is the same, except that this time we start with [RCL$_n$], then indicate the function, followed by the name of the scratch pad register. (In this case, [RCL$_n$] [÷] will cause the number in the entry register to be divided by the number in the scratch pad register you recall.)

EXAMPLES:

We want to store 2 in register 7, and 4 in register 8.

So we do this:

[2] [STn] [7]

[4] [STn] [8]

2.000	↓ 7
4.000	↓ 8

Now we can use the numbers stored in these registers to operate on numbers in the entry register.

8 8 8 8 RCL_n ÷ 8	2222.000 ÷ ↑ 8
RCL_n X 7	4444.000 X ↑ 7
RCL_n - 7	4442.000 - ↑ 7

Note the printed symbols on the right-hand side of the tape which tell you exactly what you did.

One Final Note About Register Arithmetic

When doing arithmetic operations into a scratch pad register, the results will be stored in that register. The entry register is unchanged and any algebraic operation in progress is unaffected. If the register being operated on overflows, its contents will be meaningless, and the machine will go into OVERFLOW condition. When doing arithmetic operations out of a register, the results will be stored in the entry register and may be used in an algebraic sequence. The scratch pad register is unchanged. If the entry register overflows, its contents will be meaningless, and the machine will go into OVERFLOW condition. (Remember that the 445 overflows when it tries to work with numbers outside the range of 10^{-98} to 10^{+98}.)

MAIN DATA REGISTERS

In addition to the ten scratch pad registers we've been talking about, there are up to 512 main data storage registers (depending upon your specific machine) which are also directly accessible from the keyboard.

Each of these registers can be used in exactly the same way as a scratch pad register. But there are some differences in the procedure. Primarily because there are more main data registers to name individually. Each of the registers has a numeric name, from 00 through 511. And these numeric names are used to access each register separately. To the left of the top row of keys, there are two keys called [ST nn] and [RCL nn]. These work just like the STORE and RECALL keys we used for the scratch pad registers, except you'll notice that instead of "n", each of these keys has "nn". This is because main data storage registers are accessible only by two-digit numeric code names, instead of the single-digit names of the scratch pad registers. Main data registers 00 through 99 are accessible by pressing either of these two keys (depending upon whether you're storing or recalling) and the two-digit name of the register. But when you get to register 100, there aren't enough places to accommodate the third digit. So we have to use a hundreds code for registers 100 through 199. For this we use the DECIMAL POINT. So if we want to store a number in a register between 100 and 199, we press [ST nn] then [.] and then the last two digits of the numeric name. Similarly, for registers 200 through 299, we use [CHG SIGN] to stand for two hundreds. And we press the CHANGE SIGN key directly after the STORE or RECALL key. For registers 300 through 399, we use [EXP] to indicate three hundreds. Registers

400 through 499 use [π e] as the four hundreds code. Registers 500 through 511 are not directly accessible. That's really all there is to it. Full register arithmetic is available in each of the main data registers, and it is performed exactly the same as with the scratch pad registers. You just have to remember that the access keys require two-digit numeric codes. And you have to remember to use the [.] for hundreds, [CHG SIGN] for two hundreds, [EXP] for three hundreds and [π e] for four hundreds. For example, if you wanted to store 25 in register 362 and then multiply it by 30, you'd do this:

[2] [5] [ST nn] [EXP] [6] [2]	25.000 ↓ 62
[3] [0] [ST nn] [X] [EXP] [6] [2]	30.000 X ↓ 62
[RCL nn] [EXP] [6] [2]	750.000 ↑ 62

Notice that the 445 prints the number of the register you've used. That is, it prints a two-digit numeral code. If the register number has three digits, only the last two digits will appear on the tape.

Constant Multipliers and Dividends

The first number entered in a multiplication or division operation can be used as a constant multiplier or dividend, respectively. Let's say you wanted to multiply 2 by a series of numbers. You'd just do this:

2 X	2.000 X
3 =	3.000 =
	6.000 *
4 =	4.000 =
	8.000 *
5 =	5.000 =
	10.000 *

Or, if you wanted to divide 2 by a series of numbers (constant dividend), you'd do this:

2 ÷	2.000 ÷
3 =	3.000 =
	.666 *
4 =	4.000 =
	.500 *
5 =	5.000 =
	.400 *

The trick is that when you enter a number and hit [=], the machine goes back and performs the previously entered operation -- using the new number as the second value. You can also get a constant divisor simply by using [1/x] (reciprocal). You do it like this:

[3] [1/x]	3.000	1/x
	.333	*
[X]	.333	X
[2] [=]	2.000	=
	.666	*
[3] [=]	3.000	=
	1.000	*
[4] [=]	4.000	=
	1.333	*

As you can see, this just takes the reciprocal and sets it up as a constant multiplier. Simple -- but effective.

To the right of the block of numeral keys we've been playing with, along with [+] [-] etc., are two more keys that come in very handy. These

are [( ] and [ )] . Parentheses on the 445 can be used very much like those you would write in a formula. You can even have one set of parentheses inside another. When using parentheses on the keyboard, however, there are a few logical rules to keep in mind.

- If you close parentheses [ )] without opening them, the machine goes into ERROR mode.
- Close parentheses [ )] acts as [=] with regard to the operation inside the parentheses.
- Two-level nesting is allowed [i.e., (())] . But if you try to open a third set of parentheses without closing the second, the machine goes into ERROR mode.

Algebraic Chaining

One of the major conveniences on the 445 is algebraic chaining. Any algebraic key may be used instead of equals to terminate a previous algebraic operation and begin a new one. In the course of lengthy operations, this eliminates quite a few steps. For example,

2 + 3 = 5

5 ÷ 4 = 1.25

1.25 X 6 = 7.50

can become:

2 + 3 ÷ 4 X 6 = 7.50

You can also combine algebraic chaining with parentheses. Here are a few examples to illustrate some typical situations involving these techniques.

$\frac{4}{(6+7)}$

Keys	Display	
4 ÷	4.000	÷
(	4.000	(
6 +	6.000	+
7)	7.000	)
	13.000	*
=	13.000	=
	0.307	*

$\frac{4+5}{(6+7)}$

Keys	Display	
4 +	4.000	+
5 ÷	5.000	÷
(	9.000	(
6 +	6.000	+
7)	7.000	)
	13.000	*
=	13.000	=
	0.692	*

$1+\left(\frac{(4+5)}{(6+7)}\right)$ 1 +	1.000	+
(	1.000	(
(	1.000	(
4 +	4.000	+
5)	5.000	)
	9.000	*
÷	9.000	÷
(	9.000	(
6 +	6.000	+
7)	7.000	)
	13.000	*
)	13.000	)
	0.692	*
=	0.692	=
	1.692	*

Repeated Equals and Repeated Arithmetic

Repeated equals eliminates the necessity to re-enter values. If you depress [=] again after terminating an arithmetic operation, the 445 will repeat the operation, using the first-entered value as a constant and operating on the value then in the entry register. For example,

3 X = 9	Notice that here we didn't have to enter the 3 again, since it was already in the entry register.
3 X = = 27	When we hit [=] the second time, 9 was in the entry register. So the machine just multiplied it by 3, the first-entered value.
3 X = = = 81	Here, by pushing [=] again, we caused the machine to multiply the 27 that was in the entry register after the last operation again by 3, the first-entered value.
3 + 2 = 5	
3 + 2 = = 8	As above, the second time we pushed [=] the machine went back and this time <u>added</u> the first-entered value to what was in the entry register.
3 + 2 = = = 11	Obviously, we did the same thing again, only this time the entry register had our 8 in it.
3 X 2 = 6	
3 X 2 = = 18	Tricky. Even though we entered an intermittent value, 2, the machine still went back when we hit the second [=] and multiplied what was in the entry register by the <u>first-entered</u> value.
3 X 2 = = = 54	The same as above, except that now the entry register had 18 in it instead of 6.

As you can see by the very first example, entering a number followed by [X] and [=] is a fast way to get the square of the number. Repeated equals only works for multiplication and addition. It has no

value after subtract and divide. (If you want to find out why, just try doing it.)

An operation similar to repeated equals is repeated arithmetic. In this case, we repeat arithmetic keys instead of [=] . Here's how it works:

3 X = 9 (Look familiar?) This is equivalent to 3^2.

3 X X = 81 This is equivalent to 3^4.

3 X X X = 6561 This is equivalent to 3^8.

What's happening is this -- the first time (3 X = 9), the single digit followed by [X] and [=] gives the first-entered value an exponent of 2. Each additional [X] doubles the exponent. Thus two [X]'s equal 3^4, three [X] 's equal 3^8, etc. That's with multiplication. With addition, rather than doubling the exponent, each additional [+] doubles the answer. Like this:

3 + = 6 (= 3 X 2^1)

3 + + = 12 (= 3 X 2^2)

3 + + + = 24 (= 3 X 2^3)

Another way to look at this is simply to recognize that each time you hit [+] the number in the entry register is added to itself. Repeated subtraction and division yield 0 and 1, respectively, and therefore have no value. Try it.

MULTIPLE FUNCTIONS

Second Function and Double-Function Keys

Many of the function keys on the 445 keyboard calculate two quantities. When you press a key that calculates two functions, one is printed and left in the entry register; the other is stored in a separate register. To get at the second function, you press [2ND FUNC] . The SECOND FUNCTION key exchanges the contents of these two registers and prints the number now in the entry register. If you press [2ND FUNC] again, the two registers will again be switched, and the quantities will be back in their original places. When you press a double-function key, the first function will go to the entry register, and the second function will wait in the second function storage register.

A good example of a double-function key is [π / e] , which you'll find second from the top in the vertical row of keys just to the left of the numeral keys. Pressing this key puts π (13 digits) into the entry register and puts e (13 digits) into the second function register. If you want to print π , push [PRINT ENTRY] . And if you want e, push [2ND FUNC] . Then if you want π back again, just push [2ND FUNC] again.

On the far-right side of the keyboard is a very useful pair of double-function keys -- [Ln / LOG] and [e^x / 10^x] . [Ln / LOG] calculates both the base-e and base-ten logarithms of the number in the entry register. The number and the $\log_e$ will be printed. The $\log_{10}$ will go into the

second function register and may be recalled by pressing [2ND FUNC] . Similarly, [e^x / 10^x] calculates the antilogarithm base-e and base-ten of the number in the entry register. The number and the base-e antilogarithm will print. The base-ten antilogarithm will go into the second function register, and may be recalled by pressing [2ND FUNC] .

Third Function

In addition to functions which calculate two quantities, there are also functions which calculate three quantities. In this case, one is printed and left in the entry register; the second is stored in the second function register and may be recalled by pressing [2ND FUNC] ; and the third quantity is stored in the third function register from which it may be recalled by [Φ_n] [3] .

When you press [Φ_n] [3] , the contents of the entry register will be exchanged with the contents of the third function register, and the new value in the entry register will be printed. If you press the two keys again, the entry register and the third function register will again exchange, the new contents of the entry register will be printed and the two numbers will be back where they were before. Note that the contents of the second function register are not affected by the operation of THIRD FUNCTION. However, if you initiate a function that calculates three quantities and then press [2ND FUNC] , the second function value will be in the entry register, and the first function value (which

was in the entry register) will be in the second function register. Now if you press [Φn] [3], the contents of the entry register (which is the second function value) will be exchanged with the contents of the third function register. The result of all this will be that the first function value will be in the second function register, the second function value will be in the third function register and the third function value will be in the entry register. All of the values may, of course, be returned to their respective registers by reversing the procedure.

ADDITIONAL FUNCTIONS

Besides the functions available with individual keys, there are ten additional functions available from the keyboard by means of [Φn]. Each of these additional functions has a single-digit numeral name, and is initiated by pressing [Φn] and then the numeral name of the function you want (as we did for THIRD FUNCTION, above). You'll find a complete list of these additional functions and their numeral names in the strip just below the top row of keys. Each individual function will be described in detail as we get to the area of calculation where it is used.

Except for INTEGER/FRACTION, which we'll describe right here since there's no other logical place to do it. Pressing [Φn] [5] will separate the number in the entry register into its integer and fraction portions. The integer portion will be printed and left in the

entry register. The fraction will be put in the second function register and may be recalled with [2ND FUNC]. Both the integer and the fraction will retain the sign of the original number.

STATISTICAL FUNCTIONS

Now we get to the statistical keys and the remaining Additional Functions -- which perform specialized statistical calculations faster than a speeding bullet. The statistical functions on the 445 are interrelated in many ways, and are used together to produce results tailored to specific needs.

Set Group and Data Storage

Data entered with the keyboard data summation keys are stored in the ten scratch pad registers we discussed earlier. Summation data for three independent or dependent groups may be kept in these registers simultaneously. [SET GROUP] is used to tell the machine which Group or Groups of independent data to use — either for accumulating data or calculating functions. Dependent data are always accumulated in the same registers regardless of the Group set.

A Group is set with [SET GROUP] followed by [1] [2] or [3]. When a Group is set, you can accumulate and delete data (n-count, ΣX, ΣX^2) and calculate standard deviation, mean, standard error of the mean and z-statistic -- all with respect to that Group. Setting

Group 1, 2 or 3 also determines which two Groups are used in the t-statistic and two-variable linear regression and line calculations.

SET GROUP followed by the digit 4 causes linear regression and Line functions to operate on a three-variable basis rather than the normal two-variable basis. Other group-dependent functions initiated while Group 4 is set will automatically use the Group that was in operation just before Group 4 was set. Setting Group 1, 2 or 3 will remove the Group 4 setting and return linear regression and Line calculations to the normal two-variable basis.

We'll keep track of what data are in which register of which Group as we get to particular functions which use these Groups.

A NOTE TO BENNY:

Since Set Group data are accumulated in the ten scratch pad registers, you can enter summed data directly into their appropriate registers just as well as accumulating them with the keyboard summation keys. The function keys don't care how the data got there, just so long as they're in the right places. Then you can perform all the calculations you like on that data -- just as though they had been accumulated by the machine. (Except for three-variable data, where there's a little trick you have to use if you enter summed data. See the note following Three-Variable Linear Regression.)

When you do accumulate data with the keyboard summation keys, it's a good idea to make sure the scratch pad registers are clear before

you begin. This is done easily with [Φn] [0] which clears scratch pad registers 0-9 all at once.

Sum-Square

Using the number in the entry register as X, this key accumulates n, ΣX and ΣX^2. Depending upon which Group you've got set, [$\sum_{nxx^2}$] puts the data in these registers:

Group	Register Usage		
	n	ΣX	ΣX^2
1	1	2	3
2	4	5	6
3	7	8	9

Like this:

[SET D. P.] [2]

[SET GROUP] [1]

These operations do not print.

[Φn] [0] Clears scratch pad registers 0-9

[ADV] Separates CL from data

[2] [.] [1] [$\sum_{nxx^2}$]

[2] [.] [5] [$\sum_{nxx^2}$]

[3] [.] [1] [$\sum_{nxx^2}$]

0.00	CL
2.10	Σ 1
2.50	Σ 1
3.10	Σ 1

Note the print symbols on the right-hand side of the tape. SET D.P. and SET GROUP do not print. CL means we've cleared the scratch pad registers. The Σ means that a summation was performed on the entered value. The 1 next to the Σ means that the summation went into Set Group 1. Therefore, our sums should be in registers 1, 2 and 3, according to the chart. Let's see if they are.

RCL_n 1	number of items (n)	3.00 ↑ 1
RCL_n 2	ΣX	7.70 ↑ 2
RCL_n 3	ΣX^2	20.27 ↑ 3

As you can see, our summations are in their proper places. Now let's do another group of summations, this time into Set Group 2.

SET GROUP 2	
7 Σ_{nxx^2}	7.00 Σ 2
8 . 1 Σ_{nxx^2}	8.10 Σ 2
9 . 9 Σ_{nxx^2}	9.90 Σ 2

And, just to keep the machine honest, let's recall the registers of Set Group 2 and see what's in them.

RCL$_n$ 4	number of items (n)	3.00 ↑ 4
RCL$_n$ 5	ΣX	25.00 ↑ 5
RCL$_n$ 6	ΣX^2	212.62 ↑ 6

Everything is where it's supposed to be. Now let's do one final summation into Set Group 3.

SET GROUP 3	
1 3 Σ_{nxx^2}	13.00 Σ 3
1 1 Σ_{nxx^2}	11.00 Σ 3
1 3 . 5 Σ_{nxx^2}	13.50 Σ 3
1 4 . 1 Σ_{nxx^2}	14.10 Σ 3

And, just for consistency's sake, let's see what's in the Set Group 3 registers.

RCL$_n$ 7	number of items (n)	4.00 ↑ 7
RCL$_n$ 8	ΣX	51.60 ↑ 8
RCL$_n$ 9	ΣX^2	671.06 ↑ 9

"Grouped" Data

If you've got a whole group of identical items to enter, you don't have to enter them individually.

You can enter a group of data with this sequence:

enter X

press $\boxed{\Sigma_{nxx^2}}$

enter frequency (f) of X

press $\boxed{=}$

If f = 20, and the Set Group is 1, 20 will be added to register 1; 20X will be added to register 2; and $20X^2$ will be added to register 3.

Let's enter a group.

SET GROUP 2		
$\boxed{\bar{\Phi}_n}$ 0	Clears scratch pad registers 0-9	0.00 CL
ADV	Separates CL from data	
5 . 4 $\boxed{\Sigma_{nxx^2}}$	5.4 is entered 8 times	5.40 Σ 2
8 =		8.00 n

As you can see, the tape symbol for the frequency of X is n (i.e., the number of times X is entered). Let's see what got put in the Set Group 2 registers.

RCL$_n$ 4 number of items (n)	8.00	↑ 4
RCL$_n$ 5 ΣX (5.4 x 8)	43.20	↑ 5
RCL$_n$ 6 ΣX^2 (5.4^2 x 8)	233.28	↑ 6

Let's add another group on top of that one.

5 . 6 Σ_{nxx^2} 5.6 is entered 3 times	5.60	$\Sigma 2$
3 =	3.00	n

Now let's see what's in those registers.

RCL$_n$ 4 number of items (8 + 3)	11.00	↑ 4
RCL$_n$ 5 ΣX $((5.4x8) + (5.6x3))$	60.00	↑ 5
RCL$_n$ 6 ΣX^2 $((5.4^2x8) + (5.6^2x3))$	327.36	↑ 6

Our second group of data was tidily added to the first group.

Deleting Data

Also, you can remove data from a summation by using the DELETE key. Just enter the item you want removed from the summation along with

DELETE, and the n, X and X^2 will be removed from their respective registers. You can press [DELETE] before or after entering X, just so long as you've pressed [DELETE] before you press [Σ_{nxx^2}]. To remove a group of identical data, use the same sequence as you would use for entering the data, but press [DELETE] before you get to [=].

Here's an example of a Sum-Square sequence using the keys we've been talking about:

[SET GROUP] [3]			
[Φ_n] [0]		0.00	CL
[ADV]	separates CL from data		
[2] [.] [1] [Σ_{nxx^2}]		2.10	Σ 3
[2] [.] [3] [Σ_{nxx^2}]		2.30	Σ 3
[2] [.] [3] [4] [Σ_{nxx^2}]	2.34 is entered 4 times	2.34	Σ 3
[4] [=]		4.00	n
[2] [.] [6] [Σ_{nxx^2}]		2.60	Σ 3
[8] [.] [6] [Σ_{nxx^2}]		8.60	Σ 3

2 . 2 5 Σ_{nxx^2}	2.25 is entered 17 times	2.25 Σ 3
1 7 =		17.00 n
8 . 6 DELETE Σ_{nxx^2}	8.6 is removed from summation	8.60 Σ - 3
2 . 2 5 Σ_{nxx^2}	11 of the 2.25's we entered before are removed here.	2.25 Σ 3
1 1 DELETE =		11.00 n -

Note the print symbols for summation and deletion. When you use Σ_{nxx^2} to add data, the print symbol is Σ. When you use DELETE, the print symbol is $\Sigma-$. Also, the tape will advance a space automatically when you enter or delete a group of data.

Now let's take a look into these Set Group 3 registers.

RCL_n 7	number of items (n)	14.00 ↑ 7
RCL_n 8	ΣX	32.29 ↑ 8
RCL_n 9	ΣX^2	74.64 ↑ 9

If you add up the items we entered, and remove the ones we removed, you'll discover that our little machine is keeping it all quite orderly.

Standard Deviation, Mean and Standard Error Of The Mean (n-1 version)

The SD/MEAN key calculates the standard deviation, mean and standard error of the mean for the Set Group which is active. This function uses the following formulas:

$$SD = \sqrt{\frac{\Sigma X^2 - \frac{(\Sigma X)^2}{n}}{n - 1}}$$

(Note that the denominator under the radical is n-1, not n, so this key calculates the estimate of the standard deviation for this Group.)

$$\overline{X} = \frac{\Sigma X}{n}$$

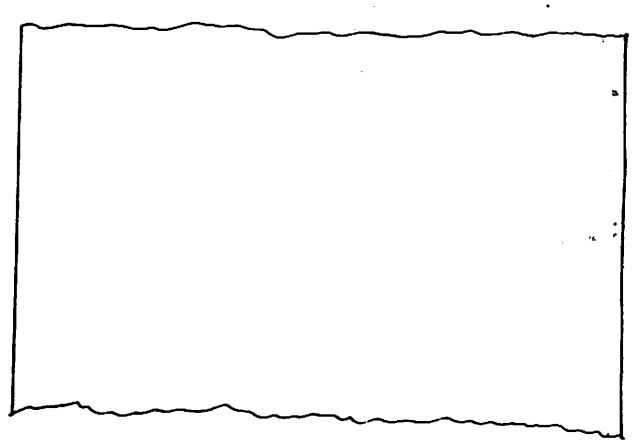
$$\text{Standard error} = \frac{SD}{\sqrt{n}}$$

The standard deviation is printed; the mean is put into the second function register and may be recalled with [2ND FUNC] ; and the standard error of the mean is put in the third function register and may be recalled with [Φ_n] [3] . No data is destroyed while performing these calculations.

Here's an example:

[SET D. P.] [3]

[SET GROUP] [3]



Σ_n 0	0.000	CL
1 4 . 5 Σ_{nxx^2}	14.500	Σ 3
1 4 . 7 2 Σ_{nxx^2}	14.720	Σ 3
1 5 . 0 1 Σ_{nxx^2}	15.010	Σ 3
1 4 . 4 4 Σ_{nxx^2}	14.440	Σ 3
1 4 . 7 2 DELETE Σ_{nxx^2}	14.720	Σ - 3
1 4 . 8 2 Σ_{nxx^2}	14.820	Σ 3
ADV		
SD MEAN takes standard deviation (estimate)	0.269	SDn-1 3
2ND FUNC gives us the mean	14.692	F 2
Σ_n 3 gives us the standard error of the mean	0.134	F 3

Note the print symbol (SDn-1) that indicates which standard deviation you've taken.

Standard Deviation, Mean and Standard Error Of The Mean (n version)

Pressing Σ_n 4 will get you the standard deviation, mean and standard error of the mean, just like SD MEAN, above, except that here the formula for standard deviation is:

$$SD = \sqrt{\frac{\Sigma X^2 - \frac{(\Sigma X)^2}{n}}{n}}$$

(Since the denominator under the radical is n (not n-1) this function is calculating the sample standard deviation.)

Using the data we've already entered in the example above, let's take the sample standard deviation, mean and standard error:

[Σn] [4]	takes standard deviation (of sample)	0.233	SDn 3
[2ND FUNC]	gives us the mean	14.692	F 2
[Σn] [3]	gives us the standard error of the mean	0.116	F 3

Note the print symbol (SDn) that indicates which standard deviation you've taken.

z-Statistic

The function [Σn] [1] calculates the z-statistic of the current Set Group. Using the number in the entry register as X, here's the formula:

$$z = \frac{X - \overline{X}}{SD}$$

Each time you press [Σn] [1], the SD and $\overline{X}$ are calculated all over again for the above formula. Here, the sample SD (not the estimate)

is used. You needn't have done a [SD MEAN] or [Φn] [4] calculation beforehand. But you do have to be sure that the Group on which you're taking a z-statistic has data in it.

The sequence you use to take a z-statistic is as follows:

enter X

press [Φn] [1]

That's all there is to it. Both X and z are printed; z is left in the entry register and may be used directly as input to the normal distribution function.

Since we've already got the right kind of data in our Group 3, and our machine is already set to that group, to get a z, all we have to do is this:

[1] [4] [.] [9] [8] [Φn] [1]

our X 14.980 X
our z 1.231 Z 3

Let's do it again with a new X:

[1] [4] [.] [9] [Φn] [1]

our X 14.900 X
our z 0.889 Z 3

Normal Distribution

[Φn] [2] calculates the area under the normal distribution curve. Using the number in the entry register as z, the total area from $-\infty$

to z (one-tailed probability) is printed and kept in the entry register. The area from -z to +z (two-tailed probability) is put into the second function register and may be recalled with [2ND FUNC]. The formula used for normal probability is

$$Y = \frac{1}{\sqrt{2\pi}} \quad (e^{-z^2/2})$$

If $z < 0$, the area from -z to +z is negative. The area in all cases is calculated to six fractional digits -- remaining digits are zeroed.

Again using the data we entered into Set Group 3 in the example under Standard Deviation (n-1) above, let's take a z and calculate the normal probability.

[1] [4] [.] [9] [Φn] [1]

[Φn] [2]

```
14.900          X
 0.889          Z    3

 0.889    P           Z
 0.813    *
```

Here, we used the z which [Φn] [1] put into the entry register as our entry for the normal probability calculation. But you don't have to do a z calculation first. You can get all the normal curve information with the entry of any z you like. Like this:

[2] [Φn] [2]

```
 2.000    P           Z
 0.977    *
```

You automatically get one-tailed probability

[2ND FUNC]

Now you get two-tailed probability

```
 0.954              F   2
```

Multiple Variable Data Entry

The XY key is used to accumulate both two-variable and three-variable data.

TWO-VARIABLE

For two-variable data, the sequence is as follows:

enter X

press [XY]

enter Y

press [=]

With this sequence, data is accumulated in these scratch pad registers:

Register	Data
0	ΣXY
1	n
2	ΣX
3	ΣX^2
4	n
5	ΣY
6	ΣY^2

Note that X data is stored in registers 1, 2 and 3 and Y data is stored in registers 4, 5 and 6. Two-variable data entry with [XY] always puts the first variable in Group 1 and the second variable in Group 2. So you can do standard deviation, z-statistic, etc. with either the X

or the Y data accumulated this way. (That's why n is stored in both register 1 and register 4.) Let's do a two-variable summation:

[Σn] [0]		0.000	CL
[ADV]	Separates CL from data		
[2] [XY]		2.000	X
[3] [=]		3.000	Y
[4] [XY]		4.000	X
[5] [=]		5.000	Y
[4] [XY]		4.000	X
[6] [=]		6.000	Y

Note the X and Y print symbols which identify each data item entered. Also notice that the tape skips a space after each XY group. Now let's take a look at what's in those registers:

RCL$_n$ 0 ΣXY	50.000	↑ 0
RCL$_n$ 1 n	3.000	↑ 1
RCL$_n$ 2 ΣX	10.000	↑ 2
RCL$_n$ 3 ΣX^2	36.000	↑ 3
RCL$_n$ 4 n	3.000	↑ 4
RCL$_n$ 5 ΣY	14.000	↑ 5
RCL$_n$ 6 ΣY^2	70.000	↑ 6

The DELETE key is used to remove data from an [XY] summation just like it was used before to remove data from a [$\sum_{nxx^2}$] summation.

Let's try removing some data from this summation:

4 XY	4.000	X
6 DELETE =	6.000	Y -

(Note the - after Y on the tape, indicating that these data have been deleted.)

And let's look at the registers again:

RCL$_n$ 0	ΣXY	26.000	0
RCL$_n$ 1	n	2.000	1
RCL$_n$ 2	ΣX	6.000	2
RCL$_n$ 3	ΣX^2	20.000	3
RCL$_n$ 4	n	2.000	4
RCL$_n$ 5	ΣY	8.000	5
RCL$_n$ 6	ΣY^2	34.000	6

The bad data are gone, and the good data live on.

THREE-VARIABLE

For three-variable data accumulation, the sequence is as follows:

enter X

press [XY]

enter Y

press [XY] again

enter Z

press [=]

With this sequence, data are accumulated in this format:

Register	Data
0	ΣXY
1	n
2	ΣX
3	ΣX^2
4	ΣYZ
5	ΣY
6	ΣY^2
7	ΣXZ
8	ΣZ
9	ΣZ^2

Here's an example of three-variable data entry:

Φ_n 0	clears scratch pad registers 0-9	0.000	CL
ADV	Separates CL from data		
2 XY		2.000	X
3 XY		3.000	Y
4 =		4.000	Z

7 XY	7.000	X
8 XY	8.000	Y
9 =	9.000	Z
1 XY	1.000	X
4 XY	4.000	Y
7 =	7.000	Z

Note that the tape skips a space after each group of three data items, just as it skips a space after each set of two items with the two-variable data sequence. And now the machine prints an X, a Y and a Z on the tape to make it easy to identify each of the data items individually.

Now to take a peek at those registers:

RCL_n 0 ΣXY	66.000	↑	0
RCL_n 1 n	3.000	↑	1
RCL_n 2 ΣX	10.000	↑	2
RCL_n 3 ΣX^2	54.000	↑	3
RCL_n 4 ΣYZ	112.000	↑	4

RCL$_n$ 5	ΣY	15.000 ↑ 5
RCL$_n$ 6	ΣY^2	89.000 ↑ 6
RCL$_n$ 7	ΣXZ	78.000 ↑ 7
RCL$_n$ 8	ΣZ	20.000 ↑ 8
RCL$_n$ 9	ΣZ^2	146.000 ↑ 9

We can, of course, remove a group of data from our summation, like this:

1 XY	1.000 X
4 XY	4.000 Y
7 DELETE =	7.000 Z -

And now the registers will look like this:

RCL$_n$ 0	ΣXY	62.000 ↑ 0
RCL$_n$ 1	n	2.000 ↑ 1
RCL$_n$ 2	ΣX	9.000 ↑ 2
RCL$_n$ 3	ΣX^2	53.000 ↑ 3
RCL$_n$ 4	ΣYZ	84.000 ↑ 4

RCL$_n$ 5	ΣY	11.000 ↑ 5
RCL$_n$ 6	ΣY^2	73.000 ↑ 6
RCL$_n$ 7	ΣXZ	71.000 ↑ 7
RCL$_n$ 8	ΣZ	13.000 ↑ 8
RCL$_n$ 9	ΣZ^2	97.000 ↑ 9

Note that when you accumulate data for three dependent variables, n is stored only in register 1 (rather than register 1 for Group 1, register 4 for Group 2 and register 7 for Group 3). For this reason, accumulation of data by the three-variable sequence sets an internal flag which tells the machine to get n from register 1 when doing SD, z and linear regression no matter which Group is set. So you can safely calculate SD and z for each of the 3 variables, even though you put the data in with the XY key and n is not in registers 4 and 7 for Set Groups 2 and 3. This flag is reset by entering data in the two-variable sequence described above, by turning the machine off, or by Σ_n 0 .

Linear Regression

LIN REG performs both two-variable and three-variable linear regression. Setting the Group to 1, 2 or 3 indicates two-variable linear regression.

Setting Group 4 indicates three-variable regression. Data for two-variable linear regression are entered as described above, using [XY] and the two-variable data sequence. Data for three-variable linear regression are entered as described above, too, using [XY] and the three-variable data sequence. Since [Σ_{nxx^2}] does not accumulate cross-products (ΣXY, ΣYZ, etc.) linear regression cannot be calculated for data entered with this key.

TWO-VARIABLE

Two-variable linear regression may be performed between any two of the three summation groups. The Group set may be 1, 2 or 3. Here's a chart that defines which two groups are used with each setting and where the data come from. (Don't forget that when you enter two-variable data with the XY key, X goes into Set Group 1 and Y goes into Set Group 2 automatically, no matter what Set Group is current. So when you do a regression on that data, Group 1 must be set.)

SET GROUP	GROUPS USED		REGISTER USAGE					
	Indep. Var. (X)	Dep. Var. (Y)	ΣXY	N	ΣX	ΣX^2	ΣY	ΣY^2
1	1	2	0	1	2	3	5	6
2	2	3	4	1	5	6	8	9
3	3	1	7	1	8	9	2	3

When [LIN REG] is pressed, the coefficients r, m, and i are calculated -- where $Y = mX + i$ is the least-square regression equation of Y on X; r is the correlation coefficient; m is the slope; and i is the intercept on the Y axis. These coefficients are calculated as follows:

$$r = \frac{\Sigma XY - \frac{\Sigma X \Sigma Y}{n}}{\sqrt{\left(\Sigma X^2 - \frac{(\Sigma X)^2}{n}\right)\left(\Sigma Y^2 - \frac{(\Sigma Y)^2}{n}\right)}}$$

$$m = \frac{\Sigma XY - \frac{\Sigma X \Sigma Y}{n}}{\Sigma X^2 - \frac{(\Sigma X)^2}{n}}$$

$$i = \frac{\Sigma X - m\Sigma X}{n}$$

When you set the Group to 1, 2 or 3 and press [LIN REG], the coefficient is printed and left in the entry register. The slope is put in the second function register and may be recalled with [2ND FUNC]. And the intercept is put in the third function register and may be recalled with [Σn] [3].

Here's an example:

[Σn] [0]

[ADV] separates CL from data

0.000	CL

1 XY	data	1.000	X
3 =		3.000	Y
1 XY		1.000	X
4 =		4.000	Y
2 XY		2.000	X
5 =		5.000	Y
1 . 5 XY		1.500	X
6 =		6.000	Y
SET GROUP 1	this guarantees that computation will use Set Groups 1 & 2		
LIN REG	calculates correlation coefficient	0.674	LR 1
2ND FUNC	gives us the slope	1.818	F 2
$\overline{\Phi}_n$ 3	gives us the intercept	2.000	F 3

THREE-VARIABLE

Three-variable linear regression is performed by setting Group 4 and then pressing [LIN REG] . Assuming, of course, that you've entered data in the three-variable sequence, as described under Multiple Variable Data Entry, above. The coefficients i, m_1 and m_2 are calculated where $Z = i_1+m_1X+m_2Y$ is the least-square regression equation of Z on X and Y. Here, i, m_1 and m_2 are determined by simultaneous solution of the following equations:

$$\Sigma Z = in + m_1 \Sigma X + m_2 \Sigma Y$$

$$\Sigma ZX = i\Sigma X + m_1 \Sigma X^2 + m_2 \Sigma XY$$

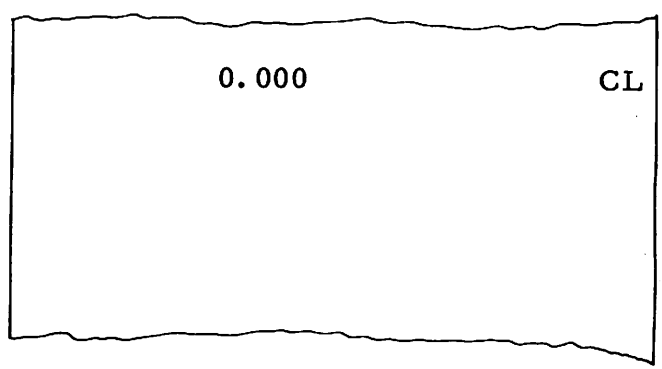
$$\Sigma ZY = i\Sigma Y + m_1 \Sigma XY + m_2 \Sigma Y^2$$

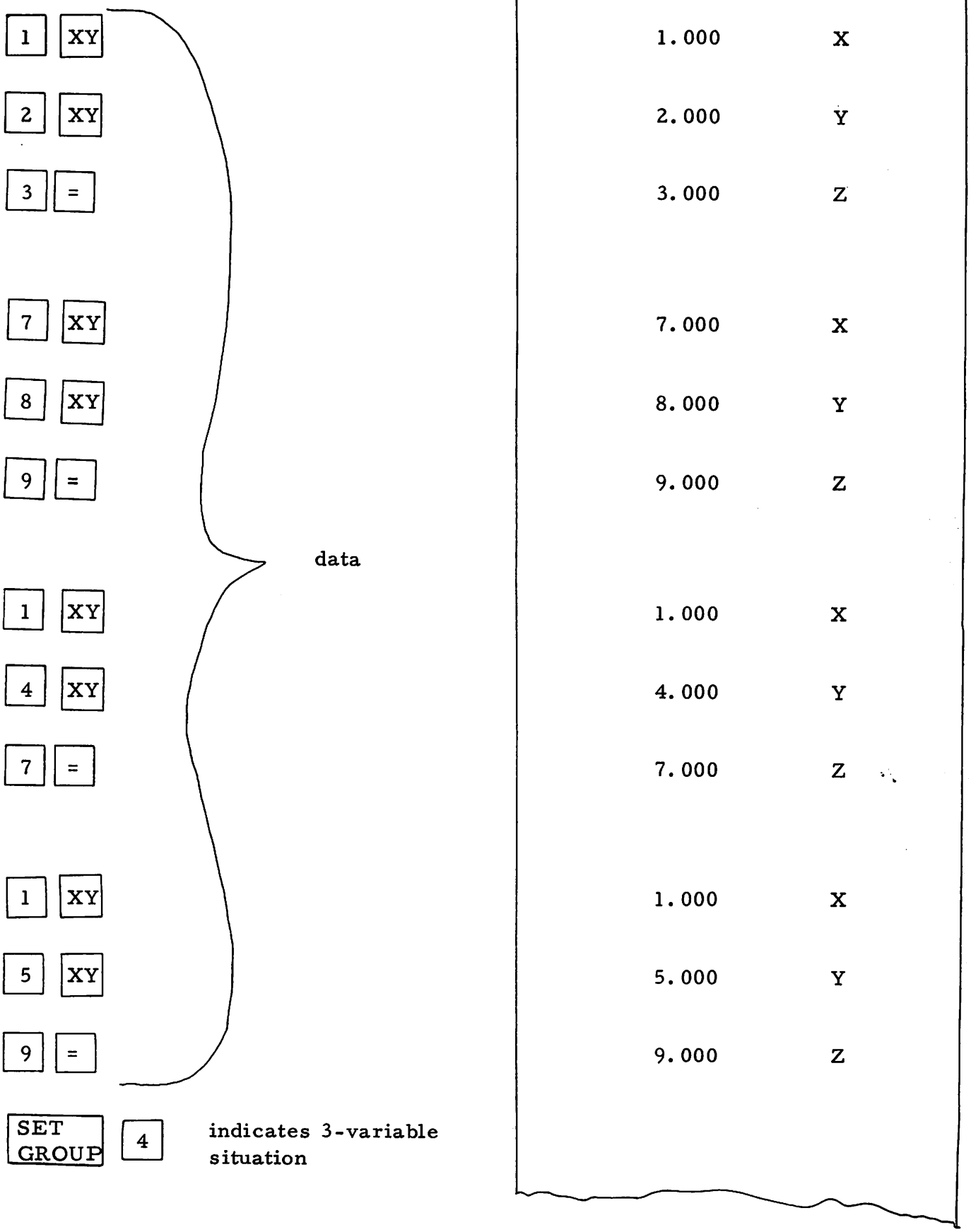
After these calculations are performed, i is printed and left in the entry register; m_1 is put into the second function register and may be recalled with [2ND FUNC] ; and m_2 is put into the third function register, from which it may be recalled with [Φn] [3] .

Let's do a three-variable linear regression to illustrate these tricks.

[Φn] [0]

[ADV] separates CL from data




1 XY
2 XY
3 =
7 XY
8 XY
9 =
data
1 XY
4 XY
7 =
1 XY
5 XY
9 =
SET GROUP 4
indicates 3-variable situation
1.000 X
2.000 Y
3.000 Z
7.000 X
8.000 Y
9.000 Z
1.000 X
4.000 Y
7.000 Z
1.000 X
5.000 Y
9.000 Z

LIN REG	i prints	
2ND FUNC	m_1 prints	$Z=i+m_1X+m_2Y$
Φ_n 3	m_2 prints	

```
0.000    LR      4
-1.000           F  2
2.000            F  3
```

We can also get the correlation coefficient, slope and intercept between any two of the three variables for which we've collected data.

Like this:

SET GROUP 1		
LIN REG	r_{xy} prints	(r between Groups 1 and 2)
2ND FUNC	m prints	Y = mX + i
Φ_n 3	i prints	
ADV		
SET GROUP 2		
LIN REG	r_{yz} prints	(r between Groups 2 and 3)
2ND FUNC	m prints	Z = mY + i
Φ_n 3	i prints	

```
0.866    LR      1
0.722            F  2
2.944            F  3
0.848    LR      2
0.960            F  2
2.440            F  3
```

ADV

SET GROUP 3

LIN REG r_{zx} prints (r between Groups 3 and 1)

2ND FUNC m prints $X = mZ + i$

Φ_n 3 i prints

0.471	LR		3
0.500		F	2
-1.000		F	3

(r = correlation coefficient)

ANOTHER NOTE TO BENNY:

As you may recall, the three-variable data entry sequence sets an internal flag to tell standard deviation, z and linear regression where to get n. If, however, you've stored the summed data directly into their respective registers, this flag never got set. Therefore the machine doesn't know where to get n for these calculations. So you have to set the flag yourself by setting Group 4 before you set the actual Group you want set for calculating.

LINE

Using the data collected for Linear Regression, this function calculates the dependent variable of either the two- or three-variable regression equation. With Group 1, 2 or 3 set, Φ_n 9 will calculate and

print Y_{est} where $Y_{est} = mX+i$. This assumes that X is in the entry register; m and i are as described under Linear Regression, above. You don't have to use [LIN REG] before using [Σn] [9] because the Line function automatically does a linear regression to determine m and i.

Here's an example:

[Σn] [0]		0.000	CL
[ADV]	separates CL from data		
[1] [XY]		1.000	X
[3] [=]		3.000	Y
[1] [XY]		1.000	X
[4] [=]		4.000	Y
	data		
[2] [XY]		2.000	X
[5] [=]		5.000	Y
[1] [.] [5] [XY]		1.500	X
[6] [=]		6.000	Y

SET GROUP 1

1 Σn 9 You give it X
It gives you Y_{est}

1.000 X
3.818 Y 1

1 . 5 Σn 9

1.500 X
4.727 Y 1

2 . 5 Σn 9

2.500 X
6.545 Y 1

With Group 4 set, the LINE function will do a three-variable Line calculation where $Z_{est} = m_1X + m_2Y + i$. Here, m_1, m_2 and i are the same as described under three-variable Linear Regression, above.

X and Y are entered like this:

enter X

press Σn 9

enter Y

press Σn 9

Z_{est} is calculated, printed and left in the entry register.

Like this:

Σ_n 0		0.000	CL
ADV	separates CL from data		
1 XY		1.000	X
2 XY		2.000	Y
3 =		3.000	Z
7 XY		7.000	X
8 XY		8.000	Y
9 =	data	9.000	Z
1 . 5 XY		1.500	X
8 . 6 XY		8.600	Y
3 . 1 =		3.100	Z
7 . 5 XY		7.500	X
4 . 6 XY		4.600	Y
1 . 8 =		1.800	Z

SET GROUP	4	indicates 3-variable situation			
1	Φ_n	9	X	1.000	X
2	Φ_n	9	Y	2.000	Y
			Z ⟶	1.645	Z 4
7	Φ_n	9	X	7.000	Z
8	Φ_n	9	Y	8.000	Y
			Z ⟶	5.938	Z 4
2	Φ_n	9	X	2.000	X
4	Φ_n	9	Y	4.000	Y
			Z ⟶	2.823	Z 4

Independent and Dependent t

This key calculates the dependent t-statistic and the independent t-statistic according to the following formulas:

$$t_{dep} = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{SD_X^2 + SD_Y^2 - 2r\, SD_X SD_Y}{n}}}$$

$$t_{ind} = \frac{\overline{X} - \overline{Y}}{\sqrt{\left[\frac{(n_X-1)\ SD_X^2 + (n_Y-1)\ SD_Y^2}{n_X + n_Y - 2}\right]\left[\frac{1}{n_X} + \frac{1}{n_Y}\right]}}$$

Where

$$\overline{X} = \frac{\Sigma X}{n_X}$$

$$\overline{Y} = \frac{\Sigma Y}{n_Y}$$

SD_X = standard deviation of X (n-1)

SD_Y = standard deviation of Y (n-1)

r = coefficient (see Linear Regression, above.)

Dependent variables should be entered with [XY] . Independent variables should be entered with [$\sum_{nxx^2}$] . The Set Group and register usage format looks like this:

SET GROUP	GROUPS USED	REGISTER USAGE						
		n_X	ΣX	ΣX^2	n_Y	ΣY	ΣY^2	n
1	1 & 2	1	2	3	4	5	6	1
2	2 & 3	4	5	6	7	8	9	1
3	3 & 1	7	8	9	1	2	3	1

As you can see, the summations and n's used are dependent upon the Group set.

With the data stored, when you press [t_{dep} / t_{ind}] the dependent t-statistic is printed and left in the entry register; and the independent t-statistic is put into the second function register and may be recalled with [2ND FUNC].

Let's do a dependent t-statistic:

[Φ_n] [0]

[ADV] separates CL from data

[1] [XY]
[3] [=]

[1] [XY]
[4] [=]

[2] [XY]
[5] [=]

paired data entry

0.000		CL
1.000	X	
3.000	Y	
1.000	X	
4.000	Y	
2.000	X	
5.000	Y	

1 . 5 XY	1.500 X
6 =	6.000 Y

SET GROUP 1	to be sure we calculate the right data	
t_{dep} t_{ind}	dependent t-statistic printed	-6.063 1 t
2ND FUNC	gives you independent t-statistic -- may be of same value	-4.539 F 2

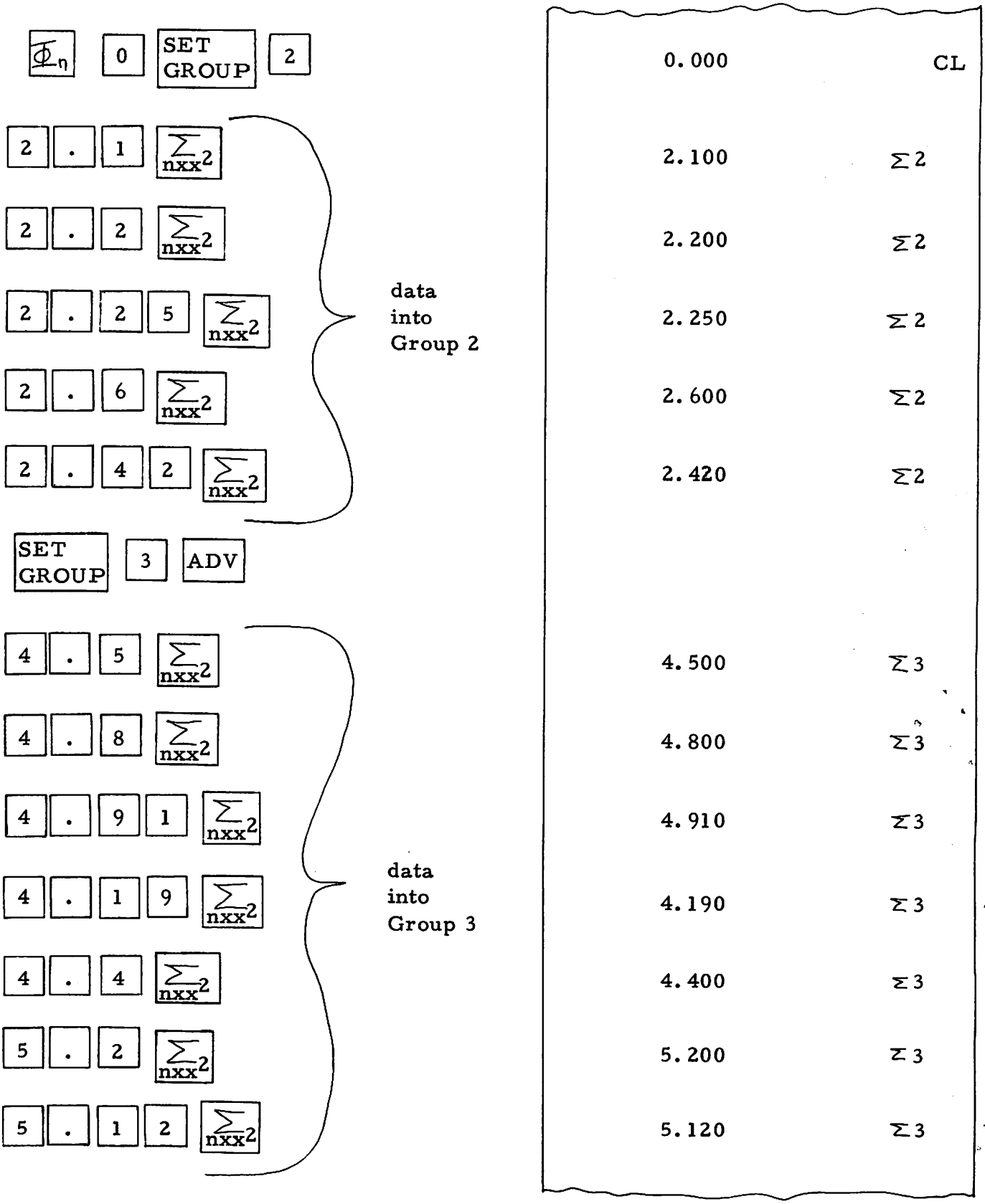
As you can see, we've entered the data with XY because the data are dependent. Note the delightful little t on the tape which identifies the dependent t-statistic. The digit before the little t indicates the Groups compared. Now let's do an independent t:

Φ_n 0 SET GROUP 1	have to set the Group where you want data to go	0.000 CL
2 . 1 Σ_{nxx^2}	independent data into Group 1	2.100 Σ 1
2 . 2 Σ_{nxx^2}		2.200 Σ 1
2 . 2 5 Σ_{nxx^2}		2.250 Σ 1
2 . 6 Σ_{nxx^2}		2.600 Σ 1
2 . 4 2 Σ_{nxx^2}		2.420 Σ 1

SET GROUP 2 ADV	set new Group, advance tape for clarity		
4 . 5 Σ_{nxx^2}	independent data into Group 2	4.500	Σ2
4 . 8 Σ_{nxx^2}		4.800	Σ2
4 . 9 1 Σ_{nxx^2}		4.910	Σ2
4 . 1 9 Σ_{nxx^2}		4.190	Σ2
4 . 4 Σ_{nxx^2}		4.400	Σ2
5 . 2 Σ_{nxx^2}		5.200	Σ2
5 . 1 2 Σ_{nxx^2}		5.120	Σ2
SET GROUP 1 ADV	must set the first of the two groups		
t_{dep} t_{ind}	meaningless here, since we've entered independent data	-0.963	1 t
2ND FUNC	here's our independent t-statistic	-12.929	F 2

This time, we've entered the data with Σ_{nxx^2} instead of XY because we have independent data. And we have to set Group 1 because our independent data are in Groups 1 and 2, and the machine will

automatically use the data in the Group set and the next group. We could have put the data in Groups 2 and 3 just as well. Like this:



SET GROUP 2

1 ST_n 1 (*)

t_{dep} t_{ind} meaningless here

2ND FUNC this is our independent t

1.000	↓	1
-0.434		2 t
-12.929	F	2

(*We must put a dummy number in register 1, since t_{dep} t_{ind} always calculates both quantities and in this case register 1, where the dependent t gets its n, will be empty unless we put something in it. If you look at the formula for t_{dep} (page 76) you'll see that n = 0 would cause the machine to go into ERROR mode.)

You can do an independent t between Groups 3 and 1, too. Like this:

Φ_n 0 SET GROUP 3

2 . 1 Σ_{nxx^2}

2 . 2 Σ_{nxx^2}

2 . 2 5 Σ_{nxx^2}

2 . 6 Σ_{nxx^2}

2 . 4 2 Σ_{nxx^2}

} data into Group 3

0.000	CL
2.100	Σ 3
2.200	Σ 3
2.250	Σ 3
2.600	Σ 3
2.420	Σ 3

SET GROUP 1 ADV		
4 . 5 Σ_{nxx^2}	4.500	Σ 1
4 . 8 Σ_{nxx^2}	4.800	Σ 1
4 . 9 1 Σ_{nxx^2}	4.910	Σ 1
4 . 1 9 Σ_{nxx^2}	4.190	Σ 1
4 . 4 Σ_{nxx^2}	4.400	Σ 1
5 . 2 Σ_{nxx^2}	5.200	Σ 1
5 . 1 2 Σ_{nxx^2}	5.120	Σ 1
SET GROUP 3 ADV		
t_{dep} t_{ind}	-1.194	3 t
2ND FUNC	-12.929	F 2

data into Group 1

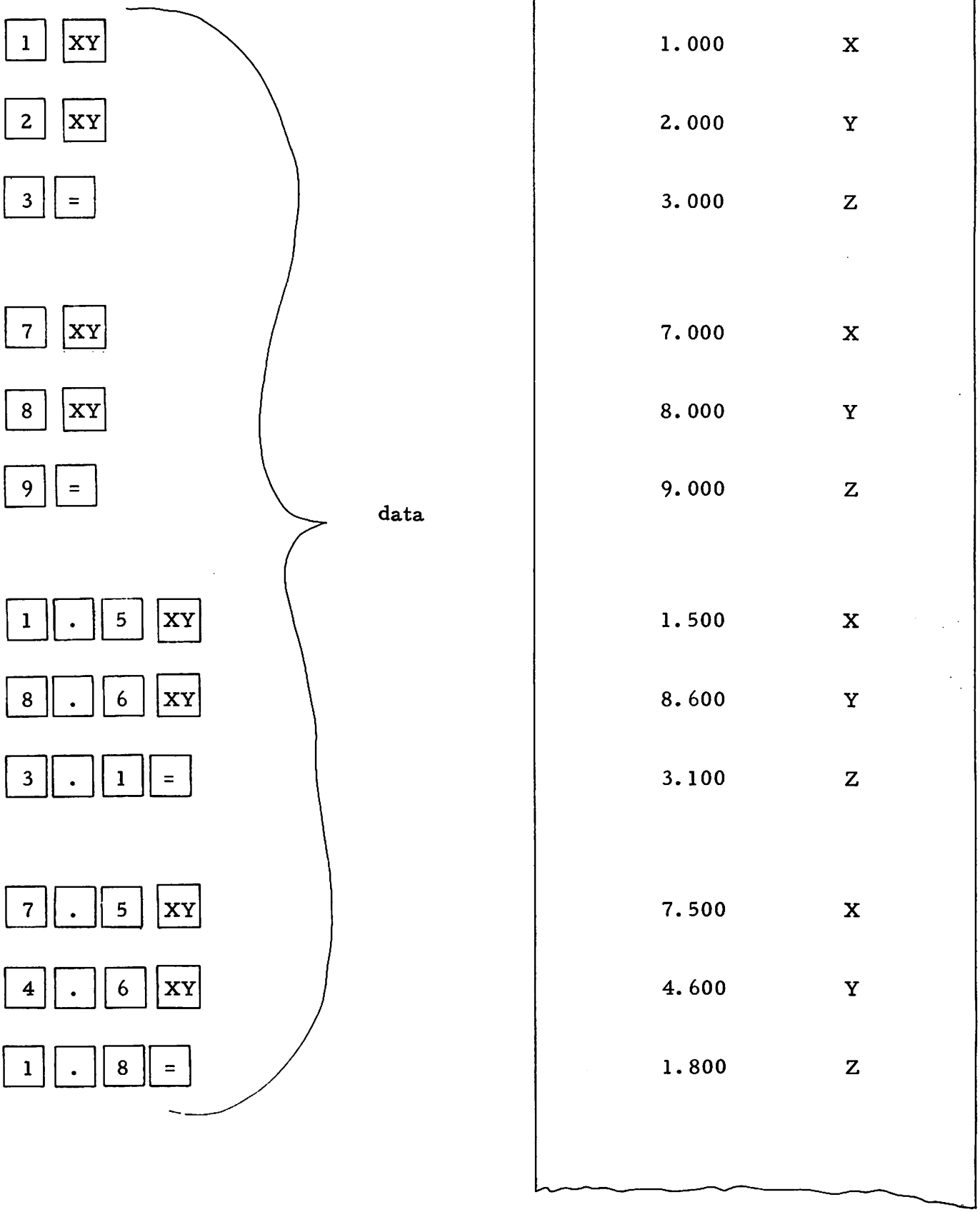
don't need a dummy number since register 1 has something in it

again meaningless here

that's our number

If you've entered three-variable dependent data, you can take t-statistics between any two of the variables. Here's how:

Φ_n 0	0.000	CL
ADV		


1 XY
2 XY
3 =
7 XY
8 XY
9 =
1 . 5 XY
8 . 6 XY
3 . 1 =
7 . 5 XY
4 . 6 XY
1 . 8 =
data
1.000 X
2.000 Y
3.000 Z
7.000 X
8.000 Y
9.000 Z
1.500 X
8.600 Y
3.100 Z
7.500 X
4.600 Y
1.800 Z

SET GROUP 1 t_{dep} t_{ind}

2ND FUNC

ADV

t-statistics for Groups 1 and 2

SET GROUP 2 t_{dep} t_{ind}

2ND FUNC

ADV

t-statistics for Groups 2 and 3

SET GROUP 3 t_{dep} t_{ind}

2ND FUNC

t-statistics for Groups 3 and 1

```
-0.750          1 t
-0.666      F   2
 0.993          2 t
 0.704      F   2
-0.013          3 t
-0.010      F   2
```

Chi-Square

The chi-square statistic is calculated with Σ_n 8 . The sequence is as follows:

enter observed frequency (f_o)

press Σ_n 8

enter expected frequency (f_e)

press =

Observed and expected values are printed; the expected value remains in the entry register. The chi-square statistic,

$$\chi^2 = \frac{\Sigma (f_o - f_e)^2}{f_e}$$

is accumulated in register 9 and the number of pairs is kept in register 7. (The machine also uses register 8 during calculation of the formula, so don't rely on scratch pad register 8 while you're using the chi-square function.)

Here's an example of chi-square:

Φ_n 0	0.000	CL
ADV		
2 Φ_n 8	2.000	o
4 =	4.000	e
3 Φ_n 8	3.000	o
7 =	7.000	e

4 Σn 8		4.000	o
8 =		8.000	e
5 Σn 8		5.000	o
12 =		12.000	e
6 6 Σn 8	oops! this should be 6, not 66	66.000	o
1 3 =		13.000	e
6 6 Σn 8	so we delete that item	66.000	o
1 3 DELETE =		13.000	e -
6 Σn 8	and put it in correctly	6.000	o
1 3 =		13.000	e

Note that the tape skips a space after each set of observed and expected data. And the print symbols o and e make it pretty easy to see which is which.

Now let's see what's in those registers:

[RCL$_n$] [7]	n-count	5.000 ↑ 7
[RCL$_n$] [9]	chi-square statistic	13.138 ↑ 9

Permutations and Combinations

[$\mathcal{I}_n$] [7] calculates the number of permutations and combinations of n items taken r at a time. The sequence is:

enter n

press [$\mathcal{I}_n$] [7]

enter r

press [=]

The values n, r and the number of permutations are printed. The number of combinations is put into the second function register and may be recalled with [2ND FUNC]. The number of permutations (P) and the number of combinations (C) are calculated according to the following formulas:

$$P = \frac{n!}{(n-r)\ !}$$

$$C = \frac{n!}{r!\,(n-r)\ !}$$

Let's do some:

8 [$\mathfrak{F}_n$] 7	8 items	8.000	n
1 =	1 at a time	1.000	r
	permutations	8.000	P
2ND FUNC	combinations	8.000	F 2
ADV			
8 [$\mathfrak{F}_n$] 7	8 items	8.000	n
2 =	2 at a time	2.000	r
	permutations	56.000	P
2ND FUNC	combinations	28.000	F 2
ADV			
8 [$\mathfrak{F}_n$] 7	8 items	8.000	n
5 =	5 at a time	5.000	r
	permutations	6,720.000	P
2ND FUNC	combinations	56.000	F 2

ADV		
8 Φn 7	8 items	8.000 n
8 =	8 at a time	8.000 r
	permutations	40,320.000 P
2ND FUNC	combinations	1.000 F 2

FACTORIAL

One last Additional Function is Φn 6 , which calculates the factorial (!) of the number in the entry register. Both the number and its factorial will print. If the number is non-integer or negative, ERROR will occur. Factorials of numbers greater than 69 are beyond the capacity of the machine (10^{98}), and you'll get OVERFLOW if you attempt one. The factorial of zero will come up as 1.

OPTIONAL KEYS

The 445 keyboard is fitted with spaces for three optional keys. If you like, you can have them installed on your keyboard with appropriate labels.

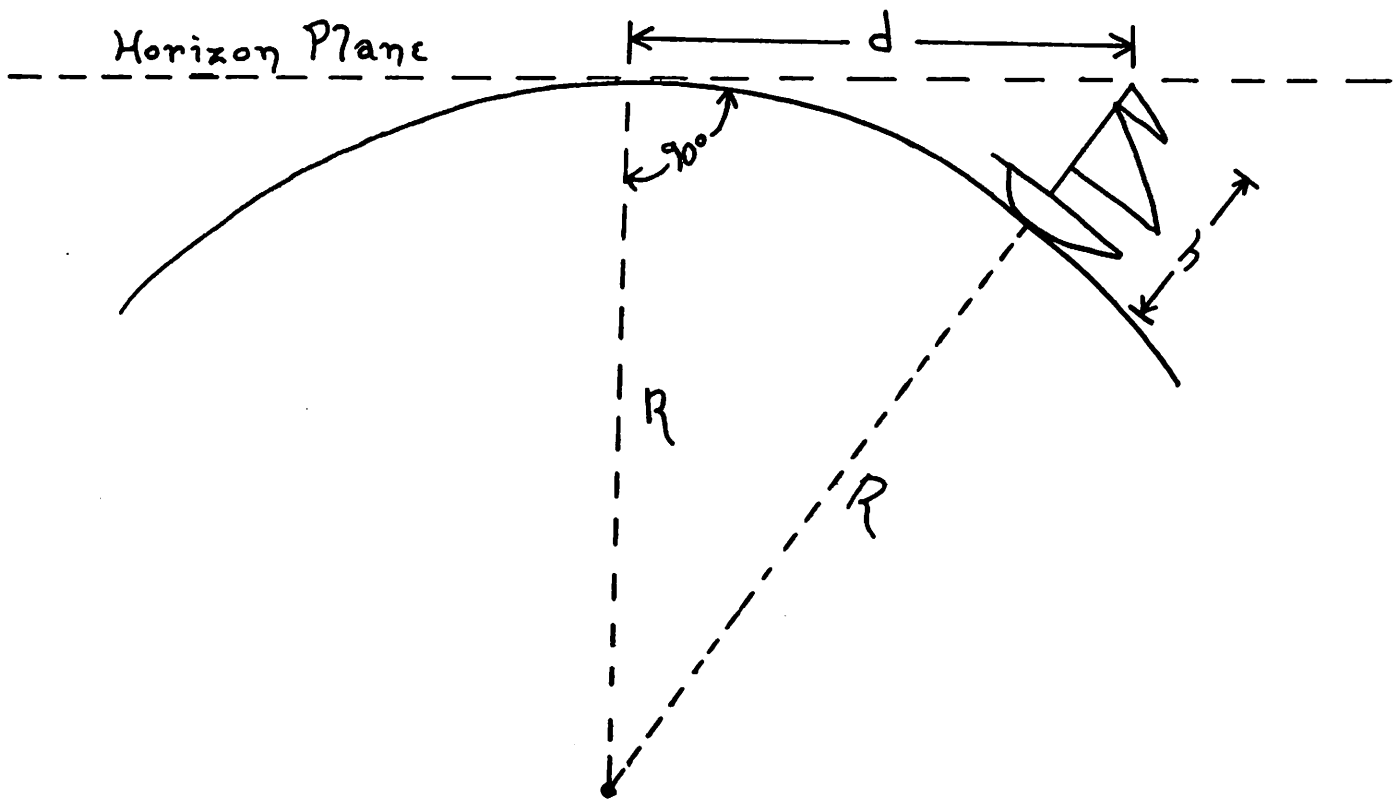
The purpose of these keys is to provide direct keyboard access to specific locations in program memory. Then if you store a program

to perform a certain operation at one of these locations you can use it as part of your keyboard capability. These optional keys will be explained thoroughly in Chapter 4.

CHAPTER THREE

GREEK SHIPS AND OTHER PHENOMENA

One day a Phoenician admiral and part-time statistician was gazing out over the horizon watching his ships disappear on their way to war exercises in the Mediterranean when suddenly a staggering thought struck him. He had been speculating on how long it took for the ships to drop below the horizon and disappear. And then this thoughts wandered to the curvature of the earth, which, although well known to people who depended upon their knowledge of the sea, was still a matter of almost total ignorance to the land-bound populace. In any case, he was ruminating on the possibility that there might be a connection between the curvature of the earth and the length of time it took ships to disappear, when another possibility captured his imagination. Perhaps there was a relationship between the distance from the ship to the observer when the ship disappears over the horizon and the curvature of the earth! And, indeed, there was. Since the radius would be the same where the ship was as it is where you are, if you knew the height of the ship you could calculate the distance from the tip of the mast to the center of the earth. Therefore, you know two sides of a triangle with a right angle in it. Now, didn't that Pythagoras fellow say if you knew all that you could find the length of the remaining side of the triangle? Yes, he did. So the admiral drew this drawing:



and then he did this figuring:

$$R^2 + d^2 = (R+h)^2$$
$$R^2 + d^2 = R^2 + 2Rh + h^2$$
$$d^2 = 2Rh + h^2$$
$$d = \sqrt{2Rh + h^2}$$

If $h = 35.2$ cubits
and $R = 13.95$ megacubits,

then $d = \sqrt{2 \times 13.95 \times 10^6 \times 35.2 + (35.2)^2}$

And, after two nights and three days of hand calculation, our ship-commanding friend finally arrived at the distance of 31.34 kilocubits. By this time, of course, the ships in question had already conquered their imaginary enemy and were on their way back to port.

If our admiral had had a 445, he could have solved the problem like this:

$$\left[d = \sqrt{2 \times 13.95 \times 10^6 \times 35.2 + (35.2)^2}\right]$$

SET D.P. 2		
RESET	0.00	^
2 X	2.00	X
1 3 . 9 5 EXP 6 X	13,950,000.00	X
3 5 . 2 +	35.20	+
(	982,080,000.0	(
3 5 . 2 X	35.20	X
=	35.20	=
	1,239.04	*
)	1,239.04	)
=	1,239.04	=
	982,081,239.0	*
√	982,081,239.0	√
	31,338.17	*

Now, the admiral has a fleet of ships with a whole bunch of different mast heights. And he wants to know how fast they're going on the way to battle. He knows when each one disappears by looking at his watch. And he can calculate the distances as above. But the statistician in him craves to know the average speed of all the ships in the fleet and the standard deviation of that speed. With a pencil and paper, this will take the admiral some time. But you can do it quite easily on the 445.

You don't really want to calculate each ship separately from beginning to end. After all, you've got a workable formula for the distance, the speed is simply the distance divided by the time and you know what the mast heights and times are. There has to be an easier way to deal with this information. And there is. You can organize your solution into a logical sequence of steps and program the 445 to do all the calculation for you. All you have to give the machine for each ship's calculation is the height of the ship from the water line to the tip of the mast, and the time it took to disappear over the horizon (we'll use $2\frac{1}{2}$ hours). Such a series might look like this:

```
2
X
1
3
.
9
5
EXP
6
```

Now, if you followed this list of steps sequentially on the keyboard, you'd wind up with the right answer (12,535.26 cubits per hour). But to get the machine to do it all without any help, you have to put in a few more steps, and

X
3
5
.
2
+
(
3
5
.
2
X
=
)
=
√
÷
2
.
5
=
Σ
nxx2

a little more information about what to do with certain pieces of data as it goes along. Don't forget that the programmable part of the 445 is exactly like a computer. (In fact it <u>is</u> a computer.) And computers can't think for themselves. You have to tell them exactly what to do at every step of the way. So, now let's go back and organize our sequence into a series of steps for <u>programming</u> the 445.

It's good to write down our program steps in the order in which we want to load them into the machine. The height of the mast will be in the entry register when the program starts (because we will have put it there), so the first thing we want the machine to do is print that number. Therefore, the first instruction is PRINT ENTRY. Now, the program doesn't really need this number at this point, so we'll store it away in a register where it will be available later. Therefore, the next two instructions will be ST_n and 0. This will put the number in register 0. Now we have to enter the time. So we program a HALT where we can enter it. Then PRINT ENTRY to print the time. And ST_n and 1

to store the time away for later use, as we did with the mast height. From here on, the program steps are the same as our sequential list, except that instead of 35.2 (the ship height in our example) we want whatever number we've got stored away in register 0, and for time we'll use the contents of register 1. The whole list of program steps, so far, looks like this:

PRINT ENTRY

ST_n

0

HALT

PRINT ENTRY

ST_n

1

2

X

1

3

.

9

5

EXP

6

X

RCL_n

0

+

(

RCL_n

0

X

=

)

=

$\sqrt{\ }$

÷

RCL_n

1

=

$\sum_{nxx}^{2}$

At this point in the program, our answer will be in the entry register. So now we want the answer printed. Our next instruction, then, is PRINT ANSWER. Then we're finished with that operation. But we know that we're going to be doing a whole series of these calculations.

We've got a HALT to enter the time. But when do we enter the next ship's height? Right now. Our next instruction, then, is HALT -- which simply tells the machine to stop and let us put something in or do something else before it goes on to the next instruction. So when we're running the program later, we'll put the next ship's height in there and it will be recorded in the entry register. But wait -- our first instruction is PRINT ENTRY. This means that if we run off a whole string of ships' heights, the heights and answers will be printed one after another on the tape without any spaces between them. That could be difficult to read. So let's put in a paper advance instruction and then a HALT instruction. Then we have to tell the machine what to do next. What we want to do is go back to the beginning of our program so we can do the calculation starting with the new number that will be in the entry register. To get there, we have to know the address where the program begins. Actually, we can start loading the program anywhere. But unless there's a reason to do it differently, programs are usually started at the beginning of program memory. So let's put ours there. The first address in program memory is 0000. To tell the machine we want to go to this address we use one of the two keys on the left side of the upper row, the [JUMP nn] or [BRANCH nn] keys, followed by [0] [0] . (The difference between BRANCH and JUMP will be explored thoroughly in Chapter 4.) Where we put our program is called the program address--since the program literally lives there.

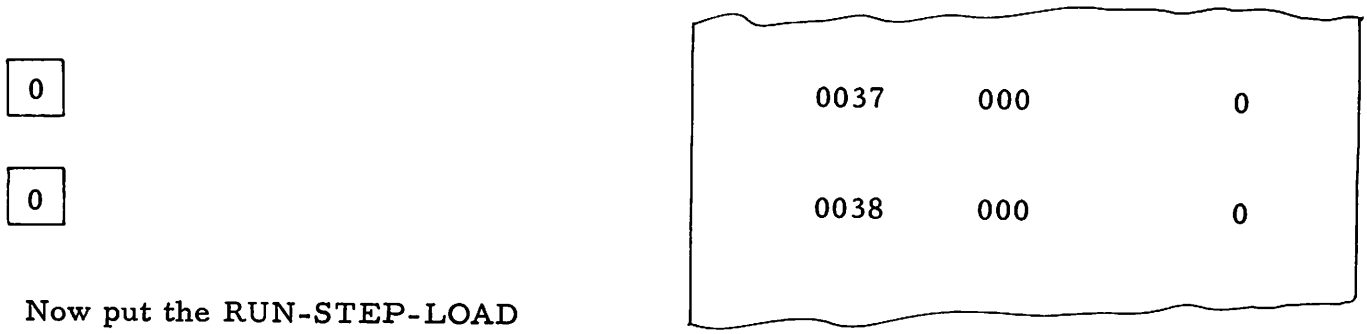
We know our program will start at 0000, so we can end our program by sending the machine back there. Our last three instructions, then, will be $JUMP_{nn}$, 0 and 0.

Now let's load our program and see if it actually works. First we have to get to the address where we're going to load it. So press [JUMP nn] [0] [0] . Then we have to tell the machine to load what we give it. So put the RUN-STEP-LOAD switch on LOAD. With this switch in LOAD position, everything you enter will be loaded into program memory in sequential order starting at the address where you start (in this case, 00). Let's put in our program.

[PRINT ENTRY]	0000	060	
[ST_n]	0001	110	↓
[0]	0002	000	0
[HALT]	0003	056	
[PRINT ENTRY]	0004	060	
[ST_n]	0005	110	↓
[1]	0006	001	1

2	0007	002	2
X	0008	023	X
1	0009	001	1
3	0010	003	3
.	0011	012	
9	0012	011	9
5	0013	005	5
EXP	0014	014	
6	0015	006	6
X	0016	023	X
RCL_n	0017	111	↑
0	0018	000	0
+	0019	021	+
(	0020	026	(
RCL_n	0021	111	↑

0	0022	000	0
X	0023	023	X
=	0024	020	=
)	0025	027	)
=	0026	020	=
√	0027	055	√
÷	0028	024	÷
RCL_n	0029	111	↑
1	0030	001	1
=	0031	020	=
$\sum_{nxx^2}$	0032	047	Σ
PRINT ANS	0033	061	A
ADV	0034	065	
HALT	0035	056	
JUMP nn	0036	126	Ju



Now put the RUN-STEP-LOAD switch back on RUN.

As you probably noticed, the 445 printed out a symbol in the right-hand column for just about everything you programmed into it. In the middle column are the numeric codes of each key you used. And the left-hand column contains the STEP number of each instruction in your program. So you can look at the tape and review your program to see if you put everything in that you wanted to. Just compare the tape to your program list. Everything in your program except PRINT ENTRY, EXP, ADV and HALT will have a symbol in the very right-hand column to identify it. If everything is the same, you're ready to run the program. But first you have to put the PRINT switch on OFF. Otherwise, the 445 will print everything it does while it's working on the program. And we really aren't interested in all of that. We just want to know what numbers we put in and what the final answer is. We've already put the RUN-STEP-LOAD switch on RUN so we don't have to worry about that. (Remember, everything you do while the RUN-STEP-LOAD switch is on LOAD is recorded in program memory.)

Let's enter our old ship's height with a time of $2\frac{1}{2}$ hours, and see if we

get the same answer we got before. Enter 35.2. Press RESUME . Enter 2.5. Then all you have to do to initiate your program is press RESUME again.

3 5 . 2 RESUME

2 . 5 RESUME

```
35.20

2.50

12,535.26        A
```

After printing out the answer, it'll stop. That's because we put a HALT there. Now we can enter the next ship's height -- say, 43.7 cubits. Then press RESUME and enter the time, say 3 hours. Now press RESUME again and the 445 will take off and do everything you've programmed it to do all over again. This time we get 11,639.16 cubits per hour. And you can keep on entering numbers and pressing RESUME until you run out of ships or your forefinger falls off.

Now let's suppose you had a more difficult problem. Let's say you want to calculate the expansion,

$$\sin X = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots$$

Let's also say that all we need is the result of the first three terms in the expansion.

Here's how you do it.

First, of course, we have to tell the 445 where to put our program. So we hit [JUMP nn] [0] [0] . Now we put the RUN-STEP-LOAD switch on LOAD and begin entering our program.

[PRINT ENTRY]	Print X	0000	060	
[ST_n] [0]	Store X	0001	110	↓
		0002	000	0
[ST_n] [1]	Start collecting values for final answer	0003	110	↓
		0004	001	1
[a^x]	$\frac{X^3}{3!}$	0005	025	a^x
[3]		0006	003	3
[÷]		0007	024	÷
[3]		0008	003	3
[Σn]		0009	116	Σ
[6]		0010	006	6
[=]		0011	020	=

ST_n			
-	Subtract $\frac{X^3}{3!}$ from answer		
1			
RCL_n			
0			
a^x			
5	$\frac{X^5}{5!}$		
÷			
5			
Φ_n			
6			
=			
ST_n			
+	Add $\frac{X^5}{5!}$ to answer		
1			

0012	110		↓
0013	022	-	
0014	001	1	
0015	111		↑
0016	000	0	
0017	025	a^x	
0018	005	5	
0019	024	÷	
0020	005	5	
0021	116	Φ	
0022	006	6	
0023	020	=	
0024	110		↓
0025	021	+	
0026	001	1	

RCL$_n$ 1	Recall answer		
PRINT ENTRY	Print answer (not rounded)		
ADV	Advance tape		
HALT	Wait for new X		
JUMP 0 0	Go back to beginning to start calculation with new X		

0027	111	↑
0028	001	1
0029	060	
0030	065	
0031	056	
0032	126	Ju
0033	000	0
0034	000	0

Now move the switch from LOAD back to RUN.

We put in PRINT ENTRY instead of PRINT ANSWER at the end because with the kind of numbers we're calculating, we really don't want a rounded answer. For the same reason, we'd better set the decimal point for five places before we do any calculating. Let's put in .1, .2 and .3 as our values for X (in radians) and see how our program works.

SET D. P. 5

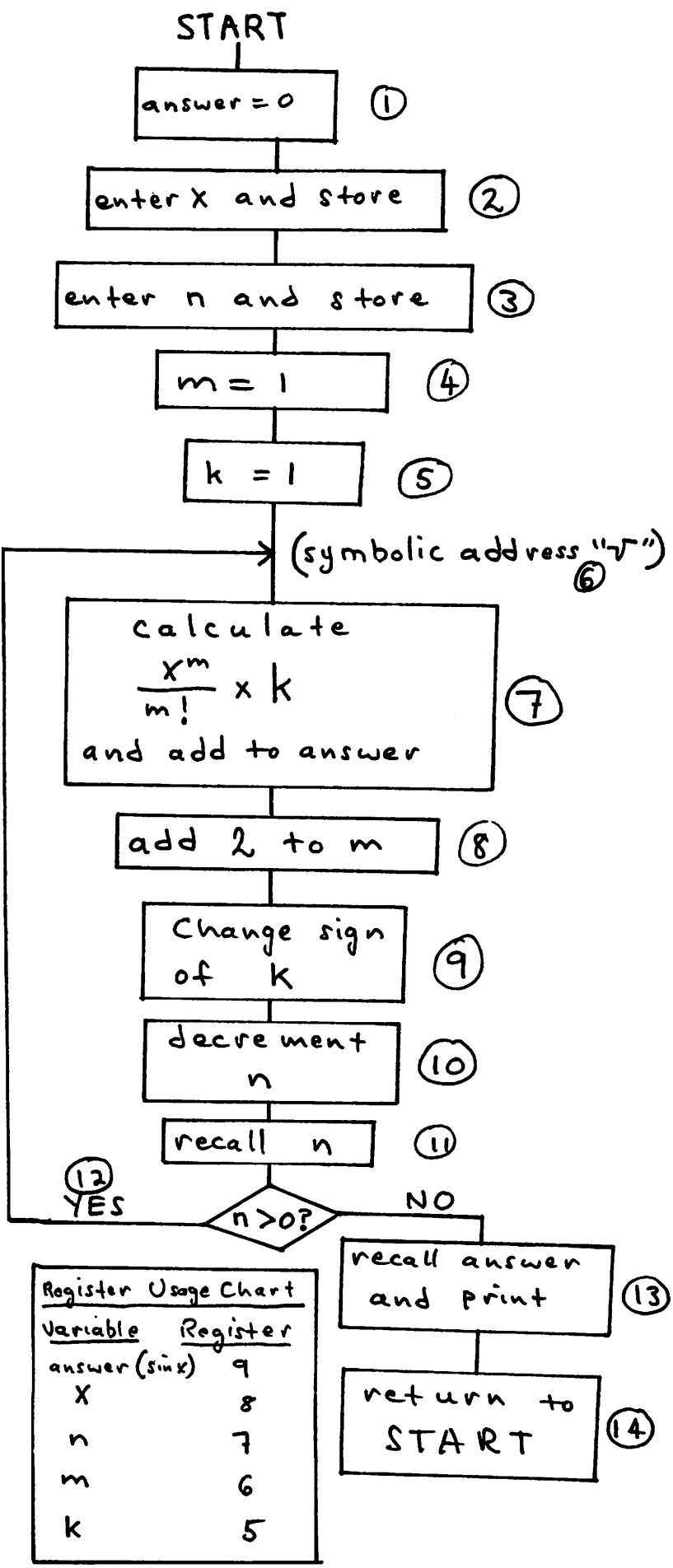
. 1 RESUME

0.10000
0.09983

. 2 RESUME	0.20000 0.19866
. 3 RESUME	0.30000 0.29552

You can see if your program is working by calculating the values on the keyboard according to the formula.

Now, let's add a bit of complexity to the program. Let's suppose that you want to continue calculating terms in the expansion for n terms. Therefore, you have to have a way of entering n, and of testing for n. A good way to start putting together a program with this degree of interdependence is to chart the flow of operations the machine will have to perform. This allows you to figure out and see clearly exactly how each element in the program relates to the other elements, and in what order they have to take place. This technique is called flowcharting, and the chart itself is called a programming flowchart. There are all kinds of different symbols that computer programmers use to indicate every little detail of their system, but for our little machine we only need two flowcharting symbols. One is a simple rectangle, in which you put operations. The other is a diamond shape, in which you put decisions. That's all you need for a flowchart. Let's make one.



Flowcharts generally run from top to bottom, left to right. As you can see, we've set up two arbitrary values, m and k, which we use to increment X and to alternate the sign of each new term. We've also set up a place where we store n. After doing the calculations we decrement n and then test it to see if it's zero. If it is, then we've done all the terms we wanted to, and we can print the answer. If not, we just go back and calculate another term, decrement n and test again.

Each block in the flowchart here has a number. You can see how we carry out each of these elements in the actual program by studying the steps next to the corresponding number. Note that we're now using a loop to calculate the terms instead of doing them sequentially.

OK. We've got a flowchart. Now let's take a look at the actual program as we load it into the machine. (Don't forget to hit JUMP nn 0 0 , and to put the machine in LOAD mode.)

RESET
ST_n
9

(1) Here, the zeroes which RESET puts in the entry register are stored in register 9. The answer will be accumulated here.

HALT
PRINT ENTRY
ST_n
8

(2) HALT to enter **X**, then print it and store it away.

HALT
PRINT ENTRY
ST_n
7

(3) HALT to enter n, then print it and store it away.

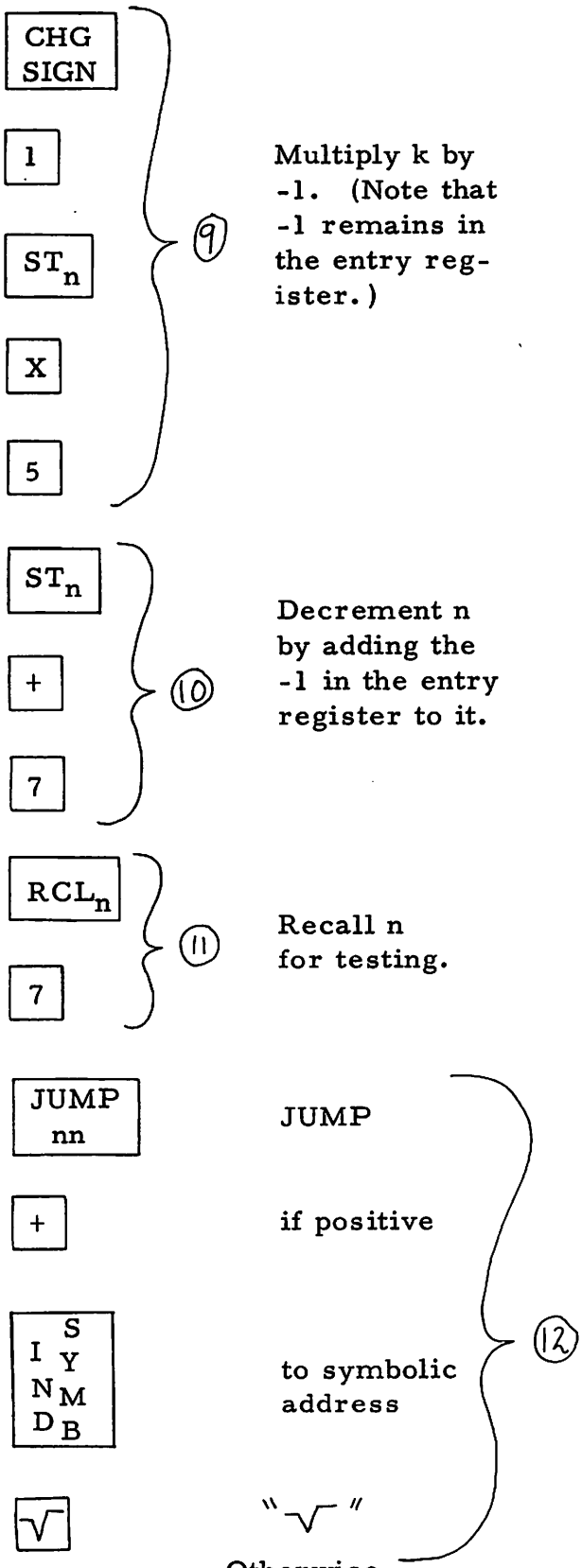
0000	062	^	
0001	110		↓
0002	011	9	
0003	056		
0004	060		
0005	110		↓
0006	010	8	
0007	056		
0008	060		
0009	110		↓
0010	007	7	

1			
ST_n	④	establish m = 1	
6			
ST_n			
5	⑤	establish k = 1	
IND SYMB		This establishes the symbolic address " $\sqrt{}$ " for this point in the program.	
$\sqrt{}$	⑥		
RCL_n			
8			
a^x			
RCL_n		Calculate $\frac{X^m}{m!} \times k$	
6	⑦		
÷			
RCL_n			
6			

0011	001	1	
0012	110		↓
0013	006	6	
0014	110		↓
0015	005	5	
0016	066		
0017	055	$\sqrt{}$	
0018	111		↑
0019	010	8	
0020	025	a^x	
0021	111		↑
0022	006	6	
0023	024	÷	
0024	111		↑
0025	006	6	

Φ_n			
6			
X			
RCL_n	⑦	Calculate $\frac{x^m}{m!} \times k$ (continued)	
5			
=			
ST_n			
+	⑦	add it to answer	
9			
2			
ST_n	⑧	increment m by 2	
+			
6			

0026	116	Φ_n	
0027	006		6
0028	023	X	
0029	111		↑
0030	005		5
0031	020	=	
0032	110		↓
0033	021	+	
0034	011		9
0035	002		2
0036	110		↓
0037	021	+	
0038	006		6



0039	013	-
0040	001	1
0041	110	↓
0042	023	X
0043	005	5
0044	110	↓
0045	021	+
0046	007	7
0047	111	↑
0048	007	7
0049	126	Ju
0050	021	+
0051	067	
0052	055	√

RCL$_n$

9

PRINT ENTRY

ADV

(13) Recall answer, print it and advance paper.

JUMP nn

0

0

(14) Go back to "START."

0053	111	↑
0054	011	9
0055	060	
0056	065	
0057	126	Ju
0058	000	0
0059	000	0

(Don't forget to put your machine back in RUN now.)

There's a new trick here. We used IND SYMB to address a location in a program by means of a symbolic address instead of a numeric code address. We'll go into that in greater detail in Chapter 4. But now let's run our program.

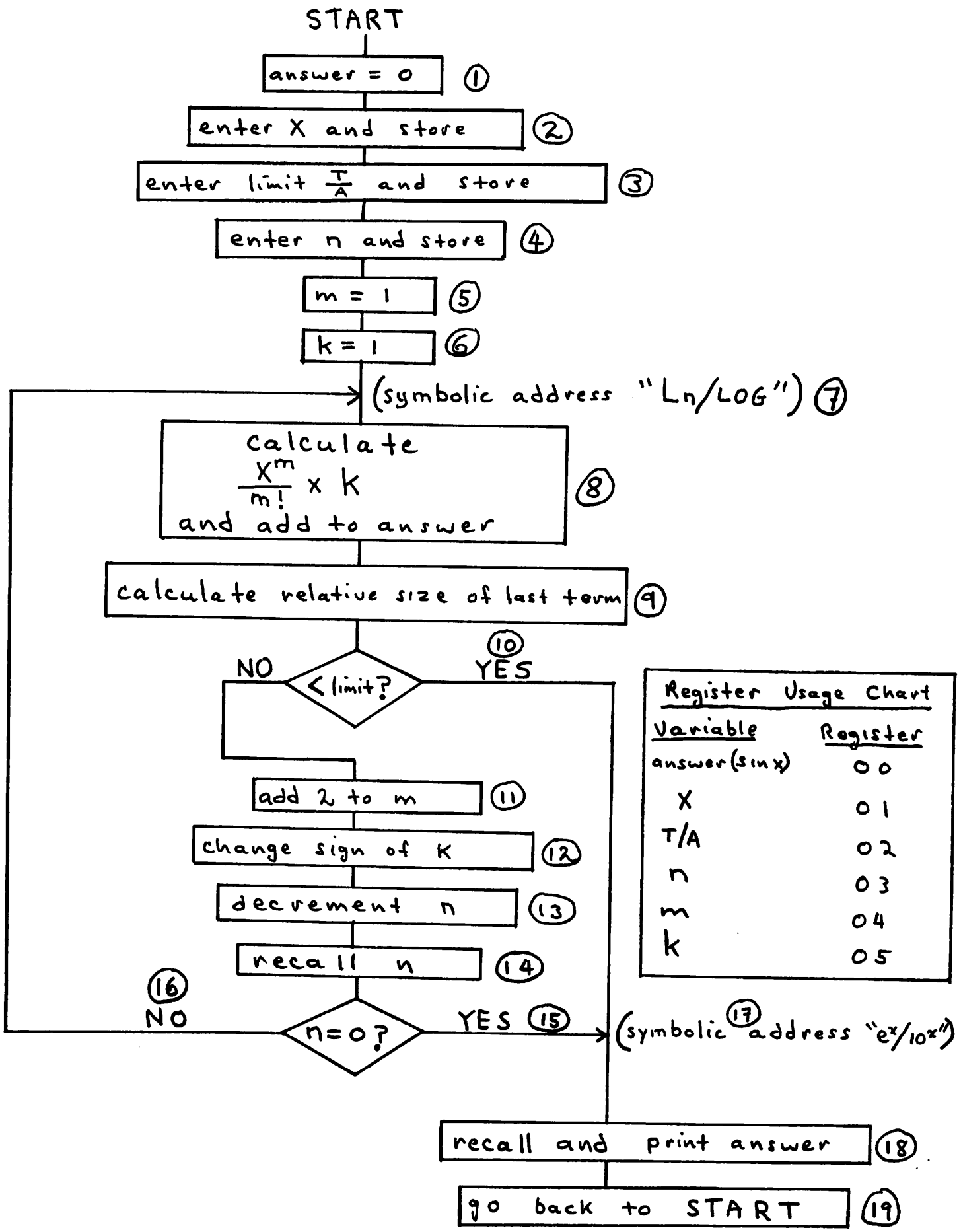
First push RESUME to get to the first HALT. Here's where you enter X. We'll use 0.3 for X. Now push RESUME again to get to the second HALT instruction. Here's where you put in n. We'll use 1 for n in the first example. Now press RESUME again. The 445

will run the program with these values and print the result. Here's a sample tape of this program using the values indicated:

X = .3 n = 1	0.30000 1.00000 0.30000
X = .3 n = 3	0.30000 3.00000 0.29552
X = .2 n = 4	0.20000 4.00000 0.19866

Now let's go on and get into a further level of complexity. Here we shall set up a program that does the same calculation, but allows us to test for two different conditions, either of which will serve as a signal to stop calculating additional terms. We'll tell the machine to stop when either a) it has calculated n terms or b) the last term calculated divided by the answer falls below a certain value. And we will make both n and the value of $\frac{\text{Term}}{\text{Answer}}$ variable.

Here's the flowchart:


START
answer = 0
1
enter X and store
2
enter limit T/A and store
3
enter n and store
4
m = 1
5
k = 1
6
(symbolic address "Ln/LOG")
7
calculate
X^m / m! x k
and add to answer
8
calculate relative size of last term
9
10
NO
< limit?
YES
add 2 to m
11
change sign of K
12
decrement n
13
recall n
14
16
NO
n=0?
YES
15
17
(symbolic address "e^x/10^x")
recall and print answer
18
go back to START
19
Register Usage Chart
Variable
Register
answer (sin x)
00
X
01
T/A
02
n
03
m
04
k
05

And here's the program as it would be entered:

RESET
ST nn
0
0

(1) The zeroes which RESET puts in the entry register are stored in register 00. This is where the answer will be accumulated.

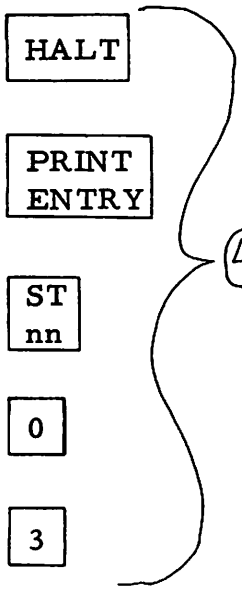
HALT
PRINT ENTRY
ST nn
0
1

(2) Enter, print and store X.

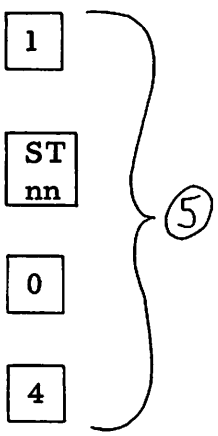
HALT
PRINT ENTRY
ST nn
0
2

(3) Enter, print and store limit.

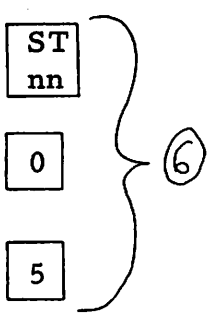
0000	062	^
0001	120	↓
0002	000	0
0003	000	0
0004	056	
0005	060	
0006	120	↓
0007	000	0
0008	001	1
0009	056	
0010	060	
0011	120	↓
0012	000	0
0013	002	2



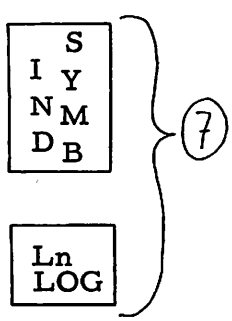
Enter, print and store n.



establishes m = 1

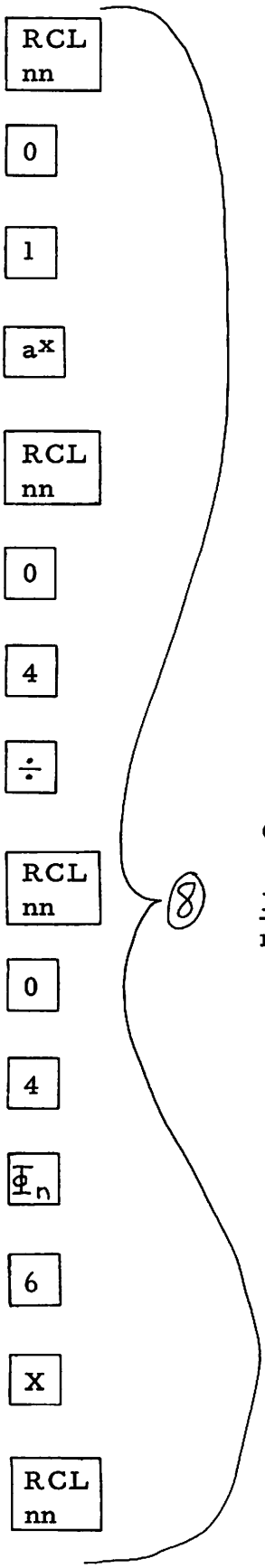


establishes k = 1



Establishes symbolic address "Ln/LOG" for this point in the program.

0014	056		
0015	060		
0016	120		↓
0017	000	0	
0018	003	3	
0019	001	1	
0020	120		↓
0021	000	0	
0022	004	4	
0023	120		↓
0024	000	0	
0025	005	5	
0026	066		
0027	050	lg	



Calculate

$$\frac{x^m}{m!} \times k$$

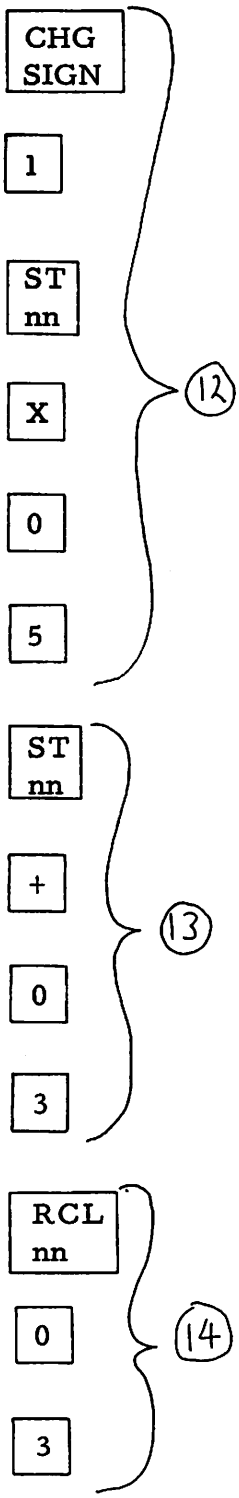
0028	121		↑
0029	000	0	
0030	001	1	
0031	025	a^x	
0032	121		↑
0033	000	0	
0034	004	4	
0035	024	÷	
0036	121		↑
0037	000	0	
0038	004	4	
0039	116	Σ_n	
0040	006	6	
0041	023	X	
0042	121		↑

0		0043	000	0
5	Calculate $\frac{x^m}{m!} \times k$ (continued)	0044	005	5
=		0045	020	=
ST nn		0046	120	↓
+	add above term to answer (note that this value will remain in the entry register.)	0047	021	+
0		0048	000	0
0		0049	000	0
÷		0050	024	÷
RCL nn		0051	121	↑
0		0052	000	0
0	$\left\lvert\frac{\text{term}}{\text{answer}}\right\rvert$ - limit	0053	000	0
=		0054	020	=
ENTER CODE nnn	0 4 5 *	0055	045	|X|
-		0056	022	-

*These steps (ENTER CODE nnn and 045) enter directly the numeric code of a programming function not accessible from the keyboard. This function is absolute value, and is used here to avoid the necessity of dealing with the sign of the number.

RCL nn			
0			
2	(9)	$\left\|\frac{\text{term}}{\text{answer}}\right\|$ - limit (continued)	
=			
JUMP nn		JUMP	
-		if negative	
SIN SYMDB	(10)	to symbolic address	
e^x 10^x		"$e^x/10^x$"	
Otherwise,			
2			
ST nn			
+	(11)	increment m by 2	
0			
4			

0057	121		↑
0058	000		0
0059	002		2
0060	020	=	
0061	126	Ju	
0062	022	-	
0063	067		
0064	051	lg^{-1}	
0065	002		2
0066	120		↓
0067	021	+	
0068	000		0
0069	004		4



change sign of k

decrement n by 1 (-1 was left in the entry register by the previous operation)

recall n

0070	013	-
0071	001	1
0072	120	↓
0073	023	X
0074	000	0
0075	005	5
0076	120	↓
0077	021	+
0078	000	0
0079	003	3
0080	121	↑
0081	000	0
0082	003	3

JUMP nn	JUMP	(15)	0083	126	Ju
=	if zero		0084	020	=
IND SYMB	to symbolic address		0085	067	
e^x 10^x	"e^x/10^x"		0086	051	lg^{-1}
Otherwise,					
JUMP nn	JUMP	(16)	0087	126	Ju
IND SYMB	to symbolic address		0088	067	
Ln LOG	"Ln/LOG"		0089	050	lg
IND SYMB	define ending point	(17)	0090	066	
e^x 10^x			0091	051	lg^{-1}
RCL nn	print answer and advance tape	(18)	0092	121	↑
0			0093	000	0

0

PRINT ENTRY

ADV

(18) print answer and advance tape (continued)

JUMP nn

0

0

(19) go back to "START"

```
0094    000          0
0095    060
0096    065
0097    126          Ju
0098    000          0
0099    000          0
```

Let's do a couple of computations on our program to see if it works.

First, let's compute:

$$X = 1$$

$$\text{limit} = 1 \times 10^{-20}$$

$$n = 3$$

SET D. P. 9

RESUME (to get to the first HALT)

1 RESUME

EXP - 2 0 RESUME

3 RESUME

```
1.000000000   00
1.000000000  -20
3.000000000   00
8.416666666  -01
```

This time, the computation stopped because of the n test. Now let's do this one:

X = 1

limit = 1 X 10^{-3}

n = 10

1 RESUME	1.000000000 00
EXP - 3 RESUME	1.000000000 -03
1 0 RESUME	1.000000000 01
	8.414682539 -01

This time, the computation stopped because of the term/answer limit.

If you lived through this last example, you should begin to get a feeling for how the 445 works. And you should begin to wonder about some of the curious tricks and subtleties of which the machine is capable. And if you really followed the last example, you should be ready to explore the further reaches of programming technique. In any case, the next chapter covers the intricate and the obvious with regard to programming the 445.

CHAPTER FOUR

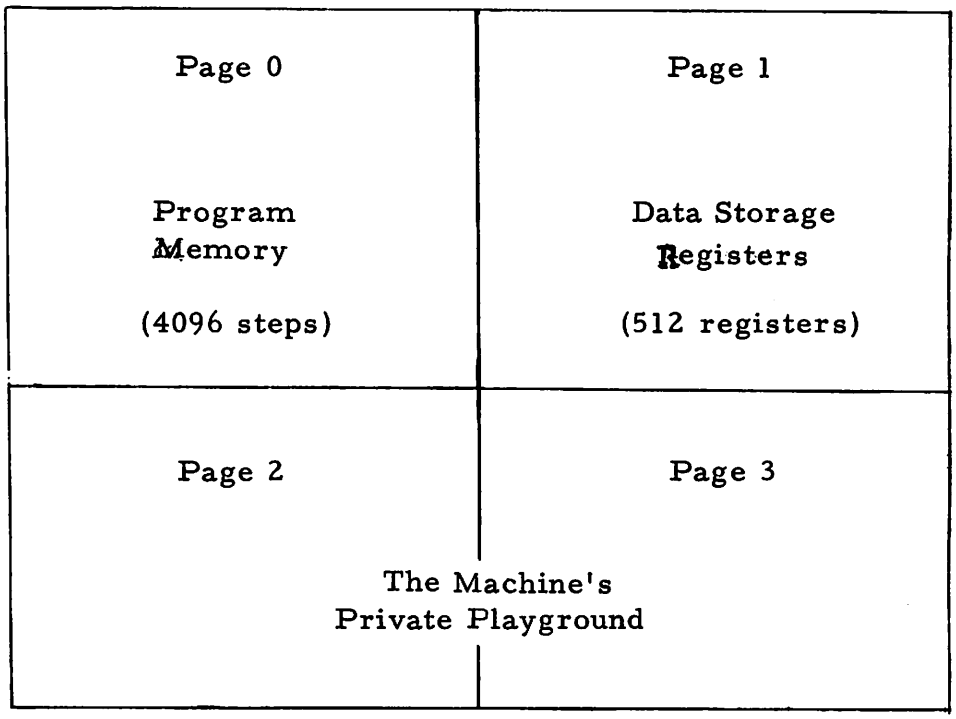
MORE ABOUT PROGRAMMING

Programming is nothing more than automatic key pushing. It is not a mysterious branch of magic which allows some people with inside information to hold exclusive domain over inanimate objects. Therefore, if you know how to tell a computer (or computer-like machine) how to do something, it'll do it. No questions asked. The 445, as we've discovered, is a little computer. Therefore, if you know how to tell it, it'll do anything for you that it's capable of doing. And there are quite a number of things it can do. You can program it to do anything for you that you can do on the keyboard. And, equally important, you can tell it by programming to do some very powerful things that you can't do on the keyboard. And that's really where this chapter comes in very handy. We are now going to go through the programming part of the 445 in very much the same kind of way as we went through the keyboard in Chapter 2. The first thing we have to do is find out where things get put when we tell the 445 what to do.

MEMORY

There are three separate memories on the 445 that you can get to directly. There are the ten scratch pad registers, which is one kind of quick-access memory. And there are up to 512 main data registers which you can also access directly and use exactly like the scratch pad registers. And also, completely separate from the scratch pad

and main data registers, there is program memory. And that's what we'll be talking about in this chapter. Let's take a look at a map of the 445's memory setup.



As you can see from the memory map, the 445's memory is divided into four separate sections, called "pages". Page 1 is where you store your numbers when you use the 512 main data registers you can get to from the keyboard. Pages 2 and 3 make up the area where the 445 has its operational memory and where it does its own figuring. That's also where the scratch pad registers are. Page 0, as it says on the map, is for program storage. The 445 stores each program instruction as a three-digit numeric code. (Later we'll discuss how you can enter these codes directly if you want to.) Program memory contains up to 4096 instruction locations. Each of these locations can hold

one instruction. Therefore, you could conceivably write a program with up to 4096 instructions and load it into your 445. As in the other memories, each data location has an address. These addresses are numbered 0000 through 4095, and are accessible in a number of ways. OK. That's memory. Now let's talk about how we get there.

DIRECT ADDRESSING

Program instruction addresses (also called program steps) are numbered decimally from 0000 through 4095. There are two ways to get to these steps. One way is SYMBOLIC ADDRESSING, which we'll get to later on. The other way, which we'll talk about here, is DIRECT (ABSOLUTE) ADDRESSING. Every tenth program step (i.e. step 0000, 0010, 0020, 0030, etc.) is a branch point, and may be addressed directly with a BRANCH or JUMP instruction. Program steps between branch points may not be addressed directly with BRANCH and JUMP alone. Since this is true, the numeric codes of branch points are shortened for access by removing the last digit -- which is always zero anyway. In addition, the first digit of these program instruction addresses is dropped if its a lowly zero, and replaced with a non-numeric code if it's 1, 2 or 3. What's left is a two-digit numeric code, which we <u>do</u> use for addressing. So program step 0000, which is the first branch point, becomes branch point 00. Program step 0050, which is the sixth branch point, is branch point 05. And

so on. Each BRANCH or JUMP instruction for direct addressing, then, must be followed by a two-digit address for branch points 000 through 099, and a non-numeric code plus a two-digit address for all other branch points. Using a non-numeric code after the BRANCH or JUMP instruction to indicate hundreds, two-hundreds and three-hundreds before entering a two-digit numeric code for the rest of the address is the same sneaky method we used for the 512 main data registers. In fact, we use the same sneaky non-numeric codes. For hundreds (branch points 100 to 199) we use the DECIMAL POINT. For two-hundreds (branch points 200 to 299) we use CHANGE SIGN. For three-hundreds (branch points 300 to 399) we use EXPONENT. Branch points beyond 399 may not be addressed by means of BRANCH and JUMP.

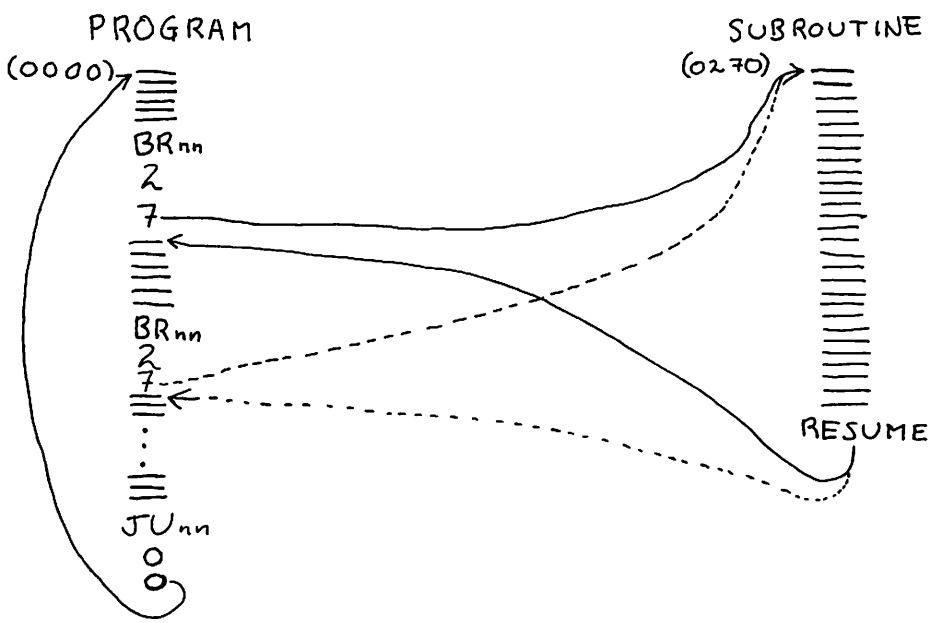
(Don't be confused by the numbering of branch points and program steps. Just remember that branch point numbers are program step numbers from which we have mercilessly removed the final zero and non-numericked the first digit.)

BRANCHING AND JUMPING

BRANCH and JUMP can be used interchangeably in many situations. But there is one major difference which must be considered. BRANCH will take you to where you designate and store in its own private place the address where you were when you branched. Then, if there's a

RESUME after the machine has done whatever you told it to do where you branched, it will return to the program step immediately following the BRANCH instruction. Therefore, BRANCH is used to enter subroutines, since you can come out of the subroutine and return directly to where you came from. JUMP, however, merely takes you to the address you indicate and drops you like a hot potato. It couldn't care less what you want to do next. If you keep this difference in mind, you shouldn't get into any trouble with BRANCH and JUMP. (There's more about subroutines later on.)

Here's a diagram of how BRANCH and JUMP work:



PRINTING

The 445 will print everything you enter while it's in LOAD mode. (And this printed record is a marvelous way of debugging and checking your program before running it.) So it doesn't matter how the PRINT switch is set while you're loading in a program. But generally you don't want the machine to print out all the intermediate steps while it's running a program. And if you have the PRINT switch ON while running your program, the machine will print out every little thing it does. So remember to include in your program PRINT ENTRY and PRINT ANSWER instructions when you want to have a number printed so you can leave the PRINT switch on OFF while running your program. As you may remember from Chapter 3, PRINT ENTRY will print whatever is in the entry register, without rounding off, while PRINT ANSWER will round off what's in the entry register. This becomes extremely important when you're dealing with very small values, where you might want to use PRINT ENTRY to print the final answer rather than PRINT ANSWER to preserve maximum accuracy out to the last digit printed.

LOADING THE PROGRAM

As you know, to load in a program, you put the machine in LOAD mode. That is, you put the RUN-STEP-LOAD switch on LOAD. When you do this, the machine will store consecutively in program memory

every key pushed. And it'll print out a record of the program you've entered. Let's take a look at that printed record and find out just exactly what all those numbers mean. To do that, let's load in a simple program and then see how the tape corresponds with what we've done. So push [BRANCH nn] [0] [0] to get to the first branch point and put the machine on LOAD. Now enter this program:

[PRINT ENTRY]	0000	060	
[X]	0001	023	X
[3]	0002	003	3
[=]	0003	020	=
[PRINT ANS]	0004	060	
[HALT]	0005	056	
[JUMP nn]	0006	126	Ju
[0]	0007	000	0
[0]	0008	000	0

Now take your machine off LOAD and put it on RUN. (It's a good idea to get into the habit of ending every program entry by taking the machine out of LOAD. That way you avoid inadvertently doing horrible things to your program while looking at the tape or talking to your secretary. You can always put it back if you need to.)

Now let's take a look at that tape. The left-hand column of numbers, the four-digit ones, is your program step number list. Each step in the program has an identifying number, starting with 0000. (This comes in handy when you're interrupted while loading a 396-step program and you forget where you were.) The next column, in the middle, contains the numeric codes of each step in the program. Every operation the 445 can perform has a three-digit code name. The code name of the numeral 2, for instance, is 002. The code name of = is 020. The last column, on the right-hand side, contains the symbol for every operation in your program that has a symbol. PRINT ENTRY and HALT, you'll notice, have no symbols. But JUMP, X and the numerals do. Most computational keys have symbols, and most purely clerical keys don't. There's a complete list in the Appendix. Now let's consider another way of getting to individual program addresses, this time using the symbols we've been talking about.

SYMBOLIC ADDRESSING

You can name any step in your program with a symbolic name. And

then you can JUMP or BRANCH directly to that location from any other location by means of the symbolic address. It makes no difference whether the numeric address of that program step is at a branch point or not. Symbolic addressing is entirely independent of branch points. You name the step by using the INDIRECT/SYMBOL key and any one of 95 symbols available. You can use as a symbol any function represented by octal code numbers 000 through 137, with the exception of 066, which is a stick-in-the-mud. (In case you've forgotten, octal numbers have no 8's or 9's.) Many of these numbers represent key codes, and may be put into your program just by pressing the key. Those which do not represent keys are entered by means of their code numbers and the ENTER CODE key. Let's say you want to give a place in your program the symbolic address " $\sqrt{\ }$ ". You do it by loading into the program [IND SYMB] and then [$\sqrt{\ }$] .

The point in the program which is defined by this symbol is the step immediately following the [$\sqrt{\ }$] . To make the program go there, all you have to do is load in [BRANCH nn] (or [JUMP nn]) then [IND SYMB] and [$\sqrt{\ }$] . Let's do one.

0000	066		
0001	055	√	
0002	060		
0003	023		X
0004	003		3
0005	020		=
0006	061	A	
0007	056		
0008	126		Ju
0009	067		
0010	055	√	

[IND SYMB] [√]

this gives the next instruction the name " √ "

[PRINT ENTRY]

[X]

[3]

[=]

[PRINT ANS]

[HALT]

[JUMP nn] [IND SYMB] [√]

this says "go to the place called '√'"

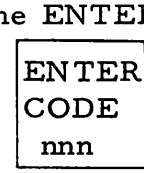
Here, we used the symbol to define the beginning of the program, and jumped back there at the end.

SYMBOLIC ADDRESS TABLE

The machine uses a table to keep track of the locations of symbols in program memory. When you name an address in program memory with a symbol, the machine puts the address of the symbol next to the symbol slot in its little table. And when you BRANCH or JUMP to that symbolic address, the machine doesn't have to look all over program memory to find it. It just takes a quick glance at its little table and goes directly to the address in program memory that it finds next to the symbol in the table. Therefore, your programs will work just as fast no matter where you put them in program memory (since the machine doesn't have to keep running through all of program memory to find things). Also, when you INSERT, the machine automatically up-dates the table. And when you read in a magnetic card (see Chapter 6), any symbols in the program(s) there are also recorded in the table -- so you can load in a symbolically addressed program anyplace in program memory.

ENTER CODE

If you want to name a place in your program symbolically with a code for which there is no key (and there are many) or if you want to load into your program a function for which there is no key on the keyboard (see numeric code list in the Appendix), you have to use the ENTER CODE key. To enter a code, all you have to do is press



and then the numeral keys for the code. (We've already done this

once. Take a look at the program on page 120.) [ENTER CODE nnn] is the only key on the keyboard which calls for a three-digit numeric entry. That makes it easy to remember that it's for three-digit numeric codes only.

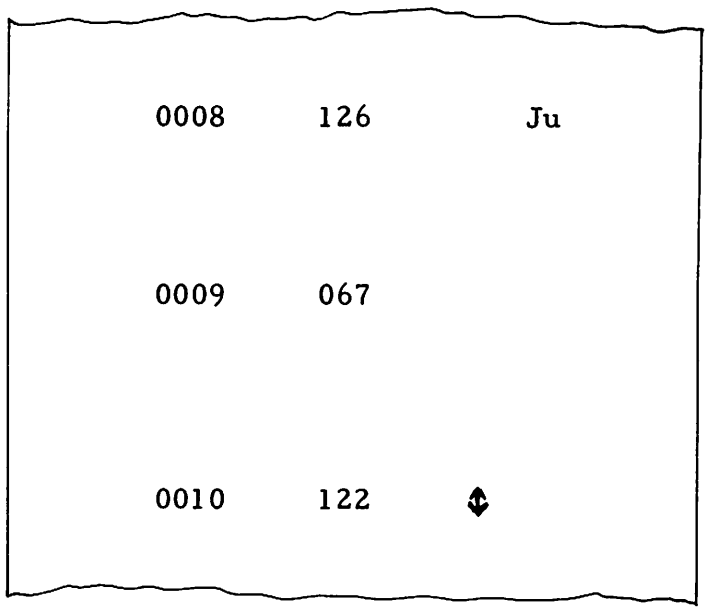
If you had wanted to use numeric code 122 instead of [√] (numeric code 055, as you can see on your tape) for your symbolic address in the example on page 135, you could have done it like this:

[IND SYMB]	0000	066	
[ENTER CODE nnn] [1] [2] [2]	0001	122	↕
[PRINT ENTRY]	0002	060	
[X]	0003	023	X
[3]	0004	003	3
[=]	0005	020	=
[PRINT ANS]	0006	061	A
[HALT]	0007	056	

[JUMP nn]

[IND SYMB]

[ENTER CODE nnn] [1] [2] [2]



As you can see from the tapes, the machine doesn't care whether you press a key or enter the numeric code. Either way, the octal code of the function you've entered will be stored in program memory.

You can BRANCH or JUMP from the keyboard to any symbolic address in a program for which there is a key on the keyboard. You do this by pressing [BRANCH nn] or [JUMP nn] then [IND SYMB] and the key which names the address. But you cannot use [ENTER CODE nnn] to get from the keyboard to a symbolic address in a program for which there is no key.

[ENTER CODE nnn] only loads numeric codes into a program -- it does not give you keyboard access to locations in program memory. Only your program can access symbolic addresses for which there are no keys on the keyboard. Similarly, you cannot use [ENTER CODE nnn] to activate a

function from the keyboard. You can only tell your program to activate this function by means of [ENTER CODE nnn]. To put it simply, [ENTER CODE nnn] is in no way a keyboard operation key. It is strictly a programming key.

BACK SPACE

A convenient key to know about while loading your program is [BACK SPACE]. Every time you press the key, you move back a space. The key is operative only in LOAD mode, and is remarkably appropriate when you make an entry mistake in the middle of your program. All you do is back up and put in the correct instruction over the incorrect one. It replaces it. If your mistake consisted of leaving out a step, then you have to use INSERT. That's next. But first, let's have a small mistake to illustrate BACK SPACE. Suppose we made a mistake while entering the program on page 135.

[SYMB IND]

[√]

[PRINT ENTRY]

```
0000    066
0001    055    √
0002    060
```

X		0003	023	X
4	(Ooops!)	0004	004	4
BACK SPACE		0003	023	X
3		0004	003	3
=		0005	020	=
PRINT ANS		0006	061	A
HALT		0007	056	
JUMP nn		0008	126	Ju
IND SYMB		0009	067	
√		0010	055	√

Yes, it works just like BACK SPACE on your typewriter. Except our BACK SPACE leaves a record of what it did.

INSERT

Here's where you get to put in a program step (or steps) that you left

out the first time. Use BRANCH, JUMP, BACK SPACE or whatever to get to the instruction just before the location where the insertion is to take place. Then press [INSERT] . The address, code and print symbol of the instruction before the insert location (the one you just got to) will be printed out, and the 445 will automatically move every subsequent instruction in program memory down one address until it gets to an empty space (a NOOP code), which it fills. Then it stops moving instructions. Check the address on the tape to be sure you're in the right place and then load in your new instruction. The IDLE light will stay off while the machine is rearranging addresses, and you may have to wait a second for it to come back on (if it has to move a lot of instructions) before you can insert another instruction. You have to press [INSERT] before each instruction you insert. If you have more than one instruction to add, load them in the same order you want them to appear in the program, pressing [INSERT] before loading in each one. Symbolic addressing is not affected by insertion because of the nifty little symbolic address table the machine keeps. When you insert, the machine up-dates the table. But it's a good idea to load several NOOP codes at the end of every program to keep the programs separated by neutral steps. This will prevent any inaccuracy in the symbolic address table due to insertion and consequent shifting of program addresses. A NOOP code (it means "no operation") just creates a very empty space. The code is 377, and you load it by pushing

ENTER CODE nnn	and	3	7	7	.

SUBROUTINES

A subroutine is simply a small program within another program. Subroutines allow you to avoid repeating an identical series of steps many times in the same program. If, for example, you have a set of instructions that perform a calculation which will take place with many items, or many times, you can set up those instructions as a subroutine -- and branch to the subroutine when you need it. This is where symbolic addressing becomes really handy, because with symbolic addresses you can load a series of subroutines one after another right after your main program. You can have subroutines within subroutines within subroutines... -- up to six levels. And remember, you BRANCH to subroutines, and BRANCH remembers where you were when you branched. RESUME gets you back there at the end of the subroutine. Let's have an example. Suppose you wanted to solve this formula:

$$Y = \frac{\sqrt{1 + LnA}}{\sqrt{1 + LnB}\ \sqrt{1 + LnC}}$$

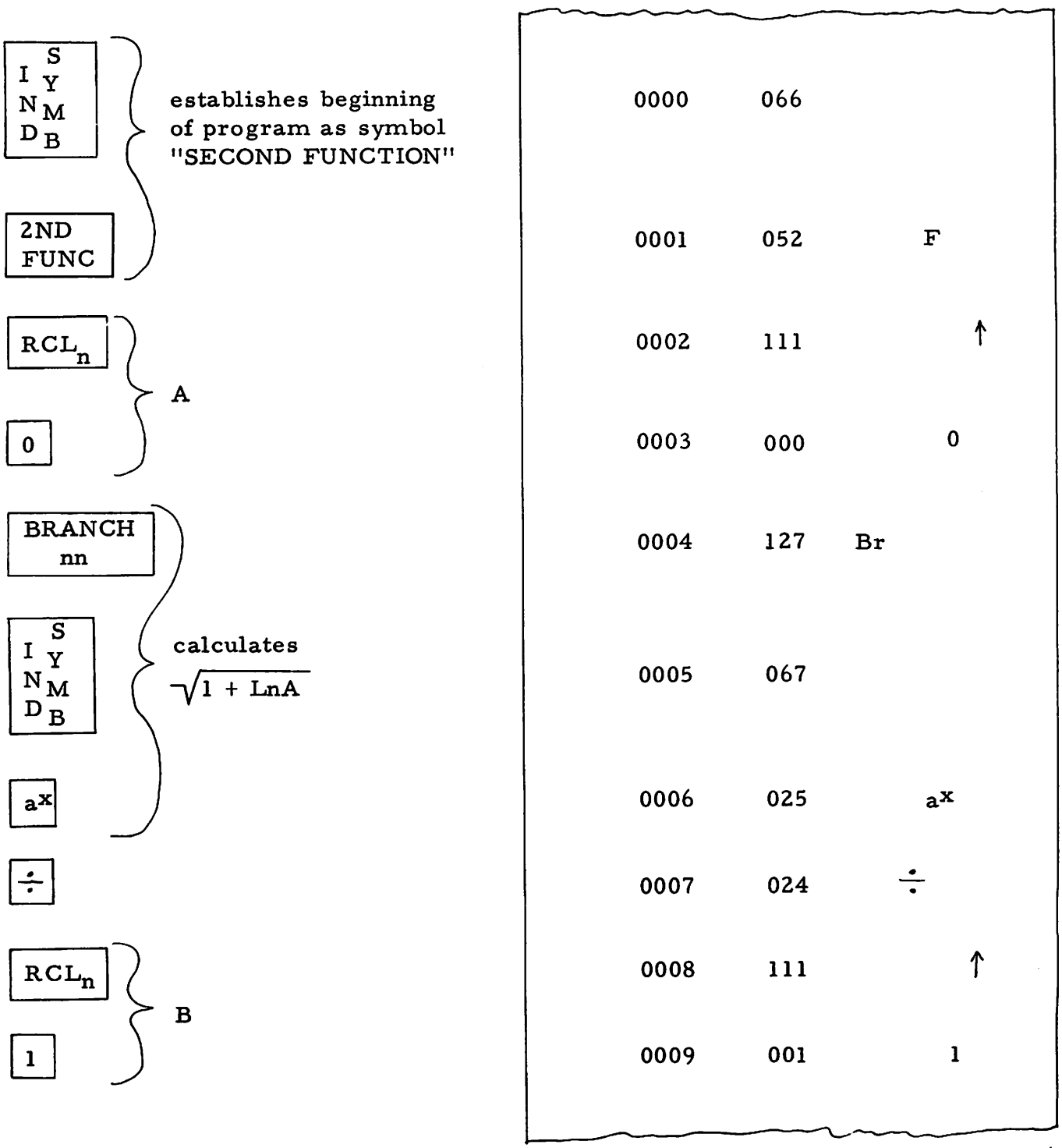
Note that $\sqrt{1 + LnA}$, $\sqrt{1 + LnB}$ and $\sqrt{1 + LnC}$ all have the same format. Therefore, we can set up a subroutine to calculate all of them rather than calculating each one separately.

Here's the program for solving this formula:

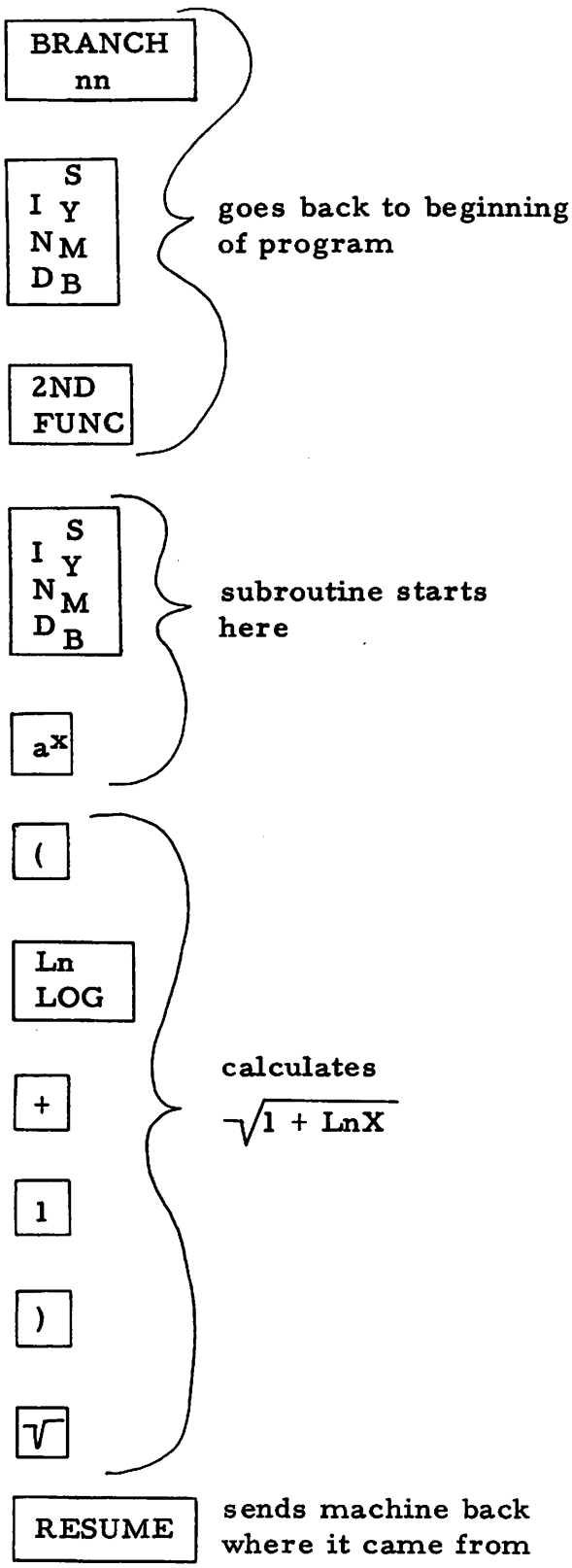
(A is in register 0

B is in register 1

C is in register 2)



BRANCH nn		0010	127 Br
INDS YMB	calculates $\sqrt{1 + LnB}$	0011	067
a^x		0012	025 a^x
÷		0013	024 ÷
RCL_n	C	0014	111 ↑
2		0015	002 2
BRANCH nn		0016	127 Br
INDS YMB	calculates $\sqrt{1 + LnC}$	0017	067
a^x		0018	025 a^x
=	calculates Y	0019	020 =
PRINT ANS	prints answer	0020	061 A
HALT	here, we could enter new values for A, B and C into registers 0, 1 and 2	0021	056



0022	127	Br	
0023	067		
0024	052		F
0025	066		
0026	025		a^x
0027	026		(
0028	050	lg	
0029	021		+
0030	001		1
0031	027		)
0032	055		$\sqrt{\ }$
0033	057		

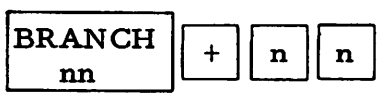
You can have as many subroutines as you have room for in program memory, as long as there is an individual address for each one. If you have several subroutines, symbolic addressing will really save you trouble -- and program memory space. You can just line them up after your program like dominoes.

CONDITIONALS

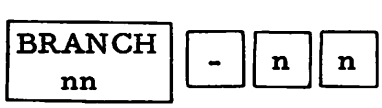
Conditionals are tests that you can build into your program. You can set up a test and have the program go one way if you get one answer, and another way if you get another answer. Each test (or condition) has two possibilities. You set it up to either BRANCH or JUMP if a condition is met. If the condition is not met, the program will just continue as if there were no test. (It's called "falling through.") When the condition is met, the program will BRANCH (or JUMP) to wherever the instruction tells it to. Here are the possible conditions on the 445 and how to set them up:

(BRANCH is used in these examples for convenience. But remember that JUMP can also be used in each case.)

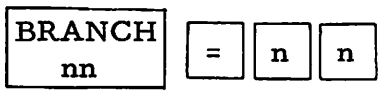
BRANCH to branch point nn if the entry register is positive.



BRANCH to branch point nn if the entry register is negative.

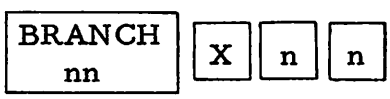


BRANCH to branch point nn if the entry register is zero.



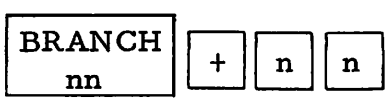
BRANCH to branch point nn if Flag 1 is set. (This is the FLAG on the keyboard.)

BRANCH to branch point nn if the OPTION switch is down.

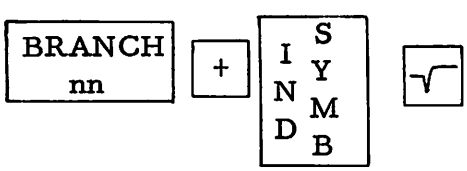


In each case, symbolic addressing can be used in place of [n][n].

So, the example



could also be



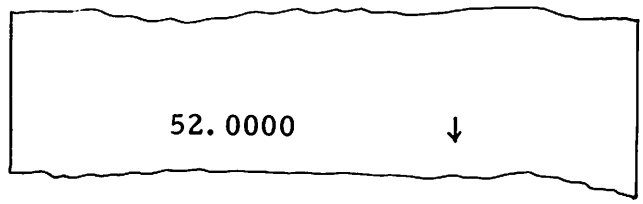
Also, remember that if RESUME is programmed at the end of the subroutine you branch to, the machine will return to the instruction directly following the second [n] (or the symbol).

INDIRECT ADDRESSING

You may remember we said earlier that there was another use for the pointer register other than as an extra scratch pad register. This is it. Register [.] can also be used to hold the address of a main data register. (It can't be used for scratch pad registers.) You just enter

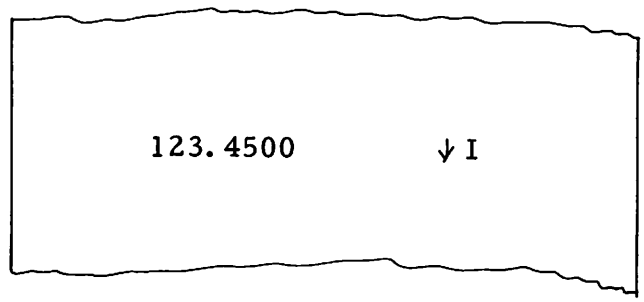
the register number you want held directly into the pointer register, say main data register 52, by

[5] [2] [STn] [.]



Now anytime you want to access register 52, you can do so indirectly by means of the pointer register. Let's say you want to store 123.45 into register 52. Just go

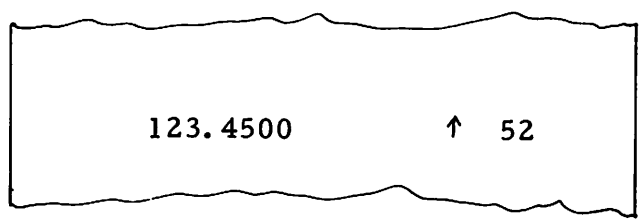
[1] [2] [3] [.] [4] [5] [STnn] [IND SYMB]



[IND SYMB] after main data STORE, RECALL or EXCHANGE instructions automatically looks into the pointer register for an indirect address.

So 123.45 is now in register 52. To prove it, you can recall register 52 directly and see what's in it:

[RCLnn] [5] [2]



You can also recall or exchange the value in register 52 indirectly by

$RECALL_{nn}$	or	$EXCHANGE_{nn}$
INDIRECT/SYMBOL		INDIRECT/SYMBOL

Of course, you can still do all the normal arithmetic operations in and out of the pointer register. This may change the number there, and

thus the indirect address. Therefore, incrementing and decrementing the address in the pointer register is extremely simple. You can access a series of registers in increasing or decreasing numerical sequence just by incrementing or decrementing the pointer and addressing indirectly.

All of the operations that can be performed in a register by means of direct addressing can also be performed in that register by means of indirect addressing. If, for example, you wanted to add 46 to register 52, you just go

[4] [6] [STnn] [+] [IND] [SYMB]

```
46.0000      +
```

Here's an example which shows the print symbols for arithmetic operations using indirect addressing.

[3] [9] [STn] [.]

[4] [0] [0] [STnn] [IND] [SYMB]

[1] [0] [5] [STnn] [+] [IND] [SYMB]

[RCLnn] [3] [9]

```
39.0000      ↓
400.0000     ↓ I
105.0000     ↓ I 1
505.0000     ↑ 39
```

2 9 8 ST_{nn} - IND SYMB	298.0000 ↓ I 2
RCL_{nn} 3 9	207.0000 ↑ 39
1 . 2 6 ST_{nn} X IND SYMB	1.2600 ↓ I 3
RCL_{nn} 3 9	260.8200 ↑ 39
2 1 ST_{nn} ÷ IND SYMB	21.0000 ↓ I 4
RCL_{nn} 3 9	12.4200 ↑ 39

Note that the print symbols are coded for arithmetic operations with indirect addressing: + is 1, - is 2, X is 3 and ÷ is 4.

SOME MECHANICAL NICETIES

Stepping Through Your Program

Bet you were wondering when we were going to get to that mysterious STEP on the RUN-STEP-LOAD switch. This is the place. With the switch on STEP, you can go through your program one step at a time. Each time you press RESUME , the address, code and print symbol for the next step are printed and the step is executed. If execution of

that step normally causes printing, then it prints. If the switch is returned to RUN, [RESUME] will cause the program to begin at the next step.

Listing Your Program

This doesn't execute anything, it just lists the address, code and print symbol for each step in your program. You can do it either one-step-at-a-time or continuously. With the switch on LOAD, press [LIST PROG] and release it immediately. This will print the address, code and symbol of the current location and stop. Press it again, and it'll do the same thing at the next location, and so on. If you hold [LIST PROG] down until the first line is printed, the entire program will be listed continuously. In fact, the entire program memory will be listed, so unless you just like to see it list, put it in RUN when you've seen enough. That'll just stop the listing. You can start it up again where it left off by putting it back in LOAD and pressing [LIST PROG] again. (You can also stop the [LIST PROG] process by pressing [HALT] . But be sure to remember you're still in LOAD mode.)

Where Am I?

If you ever wonder where you are when there's no program running, there's a way to find out. Any time the 445 is halted (IDLE light on and not flashing) the current program address can be determined by

putting the switch on either STEP or RUN and pushing [LIST PROG] . The address, code and print symbol of the current location will print. If you press [LIST PROG] again, the exact same line will print out again. And again. And again. The entry register is not changed, and no matter how many times you press [LIST PROG] , you'll stay in the same place. So now you know where you are. If the switch is on RUN and [RESUME] is now pressed, execution of the program will begin from the address printed.

Setting Flag 1

The [FLAG] key, which you'll find almost directly in the center of that top row, is a simple, direct way to set up a condition which your program can test. A program may test the flag, and conditionally BRANCH or JUMP depending on its setting. This keyboard flag is called Flag 1, and is changed only by the [FLAG] key and RESET Flag 1 instruction.

THE WONDERFUL WORLD OF ENTER CODE

We've already seen some of the things [ENTER CODE nnn] lets us do. There's more. If you press [ENTER CODE nnn] and then enter the code of the following functions, your program will do the marvelous things described below.

These programming functions and ENTER CODE work very much like the Additional Functions and [Fn] on the keyboard. Let's take them one at a time.

List-Mode Arithmetic

List-mode arithmetic uses a separate accumulator register. This accumulator is altered only by the list-mode add, subtract and total functions. It is reset to zero by RESET.

List-Mode Add — code 041

This function adds the contents of the entry register to the list-mode accumulator. The entry register is not changed. The number in the entry register is printed.

List-Mode Subtract — code 042

This function subtracts the contents of the entry register from the list-mode accumulator. The entry register is not changed. The number in the entry register is printed.

List-Mode Subtotal — code 043

This function copies the accumulator into the entry register and prints it. The accumulator is not altered.

List-Mode Total — code 040

This function totals the accumulator, copies the total into the entry register and prints that number. The accumulator is reset to zero.

Increment Entry — code 151

This instruction increments the entry register by one.

Decrement Entry — code 152

As you might expect, this instruction decrements the entry register by one.

Absolute Value — code 045

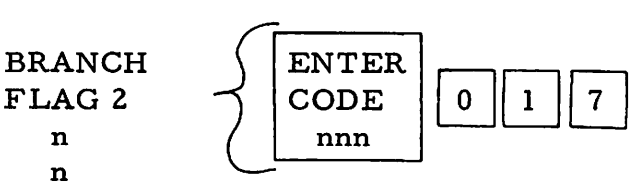
We've seen this one before. This instruction makes the sign of the entry register positive.

Square — code 053

This function calculates the square of the number in the entry register. Both the number and its square are printed.

Flag 2 — code 017

This instruction sets program Flag 2. A program may interrogate (test) this Flag and use its setting as a condition to branch or not branch. This works just like Flag 1, which you can set from the keyboard. A program for testing Flag 2 looks like this:



This translates as "BRANCH, if Flag 2 is set, to program address nn." You can use Flag 2 the same way with JUMP.

Exchange Main Data — code 122

This function exchanges data between the entry register and a main data register using the same procedure that [ST nn] uses for storage.

Dot Print — code 176

This instruction causes a line of dots to be printed, regardless of the PRINT switch position.

Equals-Sum-Zero — code 037

Equals-Sum-Zero does two things. First, it acts exactly like equals, and will produce the same result with respect to any operation you have going as would be produced by a plain [=] . But in addition, this function adds the resulting value of the entry register to scratch pad register 0. At any time, you can recall register 0 and put the accumulated total into the entry register. You can alternate between regular equals and Equals-Sum-Zero in your program all you like, keeping a running total of the values you want accumulated in one place. Remember, though, that each additional value will be added to what's already in register 0, it will not replace it. When you intend to use Equals-Sum-Zero in your program, it's a good idea to make sure register 0 is clear when you begin. There are several ways to do this, as you probably remember, including the direct procedure of storing zeroes in register 0 to begin with. The procedure you adopt in each case will depend upon what other registers you're using in your program, and whether you want to clear a series of registers or just one.

OPEN CHANNEL REGISTER ACCESS

This is another form of indirect addressing. It allows you to STORE, RECALL and EXCHANGE with a main data register by means of a fast, open channel to that register. Whenever you use the main data STORE, RECALL or EXCHANGE functions, the address of the register you've accessed is automatically kept in a separate hidden place. As long as you don't use main data STORE, RECALL or EXCHANGE instructions again, (or increment or decrement the address, see below) the address in that special place will remain the same. And there are codes for storing into the register with which this channel is open, recalling from it, and exchanging its contents with the entry register. Register arithmetic is not available with this method -- just storing, recalling and exchanging. But you can increment and decrement the address stored in that special place, and therefore access a whole series of data locations by means of the open channel method. The advantage of doing it this way is that the open channel is actually "machine level" access, and is much faster than the normal addressing procedure. The codes for using the open channel method are as follows:

Code 331 - Store the contents of the entry register into the register by the open channel pointer (the special place).

Code 332 - Recall contents of the open channel register.

Code 333 - Exchange the contents of the open channel register with the contents of the entry register.

Code 335 - Increment the address stored in the special place by 1.

Code 337 - Decrement the address stored in the special place by 1.

IDENTIFIER - Code 177

This code will cause the contents of the entry register to be printed on the very left-hand side of the tape as an identifier. Fractional digits will be included, with the decimal point in the correct position -- but trailing zeroes will be suppressed. In order to be used as an identifier, a number must have an absolute value of 1 or more.

There is a way of using the identifier code which allows an identifier to be printed while still saving the value of the entry register. If your program enters a number to be printed out as an identifier before executing code 177, that number will be printed and the former value of the entry register will be restored. If your program doesn't enter a new number before invoking code 177, the contents of the entry register will be printed as a very fine identifier, but you can't be sure of what will be left in the entry register. Only a numeral entry is considered a new number. Recalling a value from a register is not numeral entry. Here's part of a program using 2 as an identifier:

INSTRUCTION	CODE	EXPLANATION
π	015	
X	023	
=	020	π^2 now in entry register
2	002	numeral entry
IDENT	177	causes 2 to be printed as an identifier
PRINT ENTRY	060	prints π^2 , which has been restored to entry register.

With this sequence in your program, the number 2 (or any other number you use) will be printed on the left-hand side of the tape as an identifier and then abandoned. The value of the entry register just prior to your 2 (or other number) will be restored.

PRINT ENABLE - PRINT DISABLE

PRINT DISABLE (code 154) is an internal, programmable print switch. It tells the 445 not to print anything your program does except the "Print Always" functions (see Print Control, Chapter 3). PRINT DISABLE does not affect keyboard printing. It merely overrides the keyboard PRINT switch with respect to the program. Therefore, you can leave the keyboard PRINT switch ON, execute code 154 in your program, and have complete keyboard printing while your program has only "Print Always" printing capability.

PRINT ENABLE (code 155) reverses PRINT DISABLE and returns

printing control of the program as well as the keyboard to the keyboard PRINT switch.

NOTE: If you find that your program won't print, and you don't know why, it may be that someone has left the machine in PRINT DISABLE mode. To remove it, just load the instructions

155

HALT

and then execute these instructions. This will return the machine to normal PRINT control.

RECALL D.P. - code 157

This is another convenient device for the programming aspect of the 445. When you execute SET D.P., either from the keyboard or from a program, the previous decimal point setting is automatically saved. Then, when you execute RECALL D.P. (code 157) from your program, the decimal point setting is returned back to the previous setting. Where this is particularly useful is when the same machine is used for both programmed and keyboard operation. The programmer can set a decimal point that gives him the degree of accuracy he wants in his program (usually many more places than you'd want from the keyboard), and by entering code 157 at the end of his program, he returns the machine to whatever setting it had before his program began.

ADD TO REGISTER

You can add the contents of the entry register directly into any main data register simply by entering code 123 and then the address of the main data register to which you'd like the value added.

You can also add the contents of the entry register directly into any scratch pad register by entering code 113 and the numeral name of the scratch pad register.

KEYBOARD PROGRAM ACCESS

In Chapter 2, under OPTIONAL KEYS, we mentioned that there were three keys you can have added to your keyboard which provide direct keyboard access to specific locations in program memory. Since these would be your keys, you can call them whatever you like. For convenience in discussing them, though, let's call them F_1, F_2 and F_3.

Pressing [F_1] would take the machine to program memory step 495. If there was an instruction there, the machine would immediately begin executing instructions. [F_2] would take you to program memory step 500. And [F_3] would take you to program memory step 505. The purpose of these keys is to allow you to get from the keyboard directly to a program which performs some operation you don't want to do on the keyboard. With three keys, you can access three separate operations in this way. And, of course, you can change these operations any

time you want to. Since there are only five program steps at each of these locations in program memory, the logical thing to do is to use these five steps to BRANCH to some other location in program memory where you have the real program salted away.

Be sure there's a RESUME at the end of this program, so that you can return to the access area after the operation is completed. Since you got there with BRANCH, the RESUME will take you back where you came from. To return the machine to keyboard operation, there has to be a HALT instruction immediately following the BRANCH n n instructions in the access area.

If you want to use that same operation program as a subroutine in another program, just BRANCH to it from your main program. The RESUME at the end of the operation program will take you back where you branched from.

CHAPTER FIVE

EXAMPLES AND PROBLEMS

As the farmer said while he sat back and watched his wild new stallion chase his city cousin around the corral, "Showin's better'n tellin'." On that principle, this chapter consists of practical examples of how you'd go about solving certain kinds of problems on the 445. There are some pretty nifty techniques here, and it's not a bad idea to run through the examples on your own machine so you can really see how some of the subtler relationships are structured. By comparing your tape with our tape, you can also see what you did wrong if you decide to try a few out by yourself.

EXAMPLES

$$\frac{25 \times 32}{1.21 \times 15.07}$$

2 5 X	25.000 X
3 2 ÷	32.000 ÷
1 . 2 1 ÷	1.210 ÷
1 5 . 0 7 =	15.070 = 43.872 *

OR

2 5 X	25.000 X
3 2 ÷	32.000 ÷
(	800.000 (

1 . 2 1 X	1.210	X
1 5 . 0 7)	15.070	)
	18.234	*
=	18.234	=
	43.872	*

$(1 + .07)^{15}$

1 +	1.000	+
. 0 7 a^x	0.070	a^x
1 5 =	15.000	=
	2.759	*

$1 - (1 + .07)^{15}$

1 -	1.000	-
(	1.000	(
1 +	1.000	+
. 0 7 a^x	0.070	a^x
1 5)	15.000	)
	2.759	*
=	2.759	=
	-1.759	*

$$\frac{1 - (1 + .07)^{15}}{1 - .07}$$

1 -	1.000	-
(	1.000	(
1 +	1.000	+
. 0 7 a^x	0.070	a^x
1 5)	15.000	)
	2.759	*
÷	2.750	÷
(	-1.759	(
1 -	1.000	-
. 0 7)	0.070	)
	0.930	*
=	0.930	=
	-1.891	*

$$\sqrt{3 + \frac{1}{17.3}}$$

3 +	3.000	+
1 7 . 3 $\frac{1}{x}$	17.300	1/x
	0.057	*

[=]	0.057	=
	3.057	*
[√]	3.057	√
	1.748	*

$\log (3 + e^{1.2})$

[3] [+]	3.000	+
[1] [.] [2] [e^x 10^x]	1.200	lg^{-1}
	3.320	*
[=]	3.320	=
	6.320	*
[Ln LOG]	6.320	lg
	1.843	*
[2ND FUNC]	0.800	F 2

Solve:

$$Q = \frac{2}{3} b \sqrt{2g} H^{3/7}$$

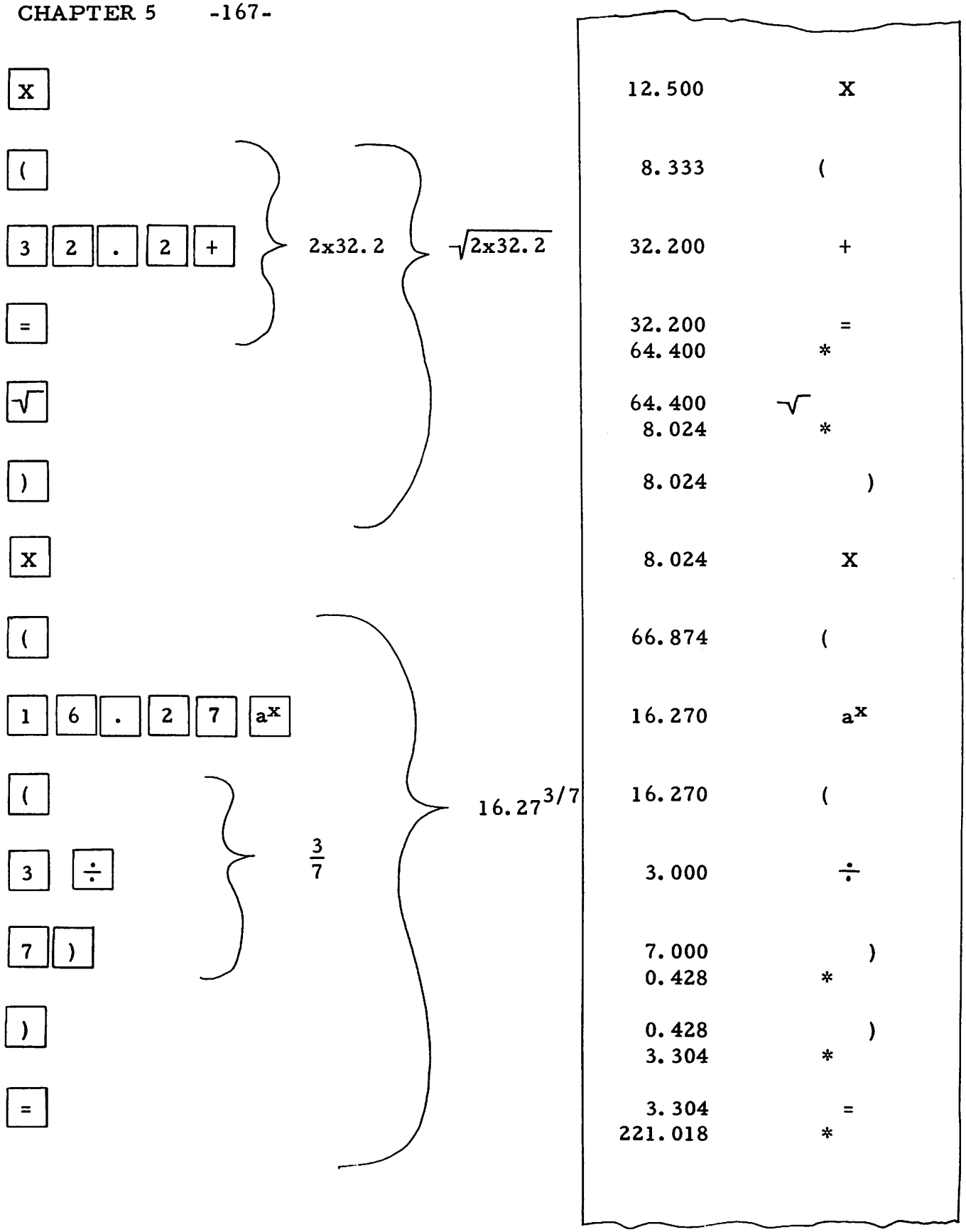
$g = 32.2$

$H = 16.27$

$b = [R_{107}]$

[2] [÷]	2.000	÷
[3] [X]	3.000	X
[RCL nn] [.] [0] [7]	12.500	↑ 07

2/3b


X
(
3 2 . 2 +
2x32.2
=
√
2x32.2
)
X
(
1 6 . 2 7 a^x
(
3 ÷
7)
3/7
16.27^3/7
)
=
12.500 X
8.333 (
32.200 +
32.200 =
64.400 *
64.400 √
8.024 *
8.024)
8.024 X
66.874 (
16.270 a^x
16.270 (
3.000 ÷
7.000)
0.428 *
0.428)
3.304 *
3.304 =
221.018 *

PROBLEMS

Problem 1

Given the following group of scores:

21.7

28.5

31.2

24.6

27.0

38.1

23.2

A. Compute the standard deviation, mean and standard error of the mean for the group.

SET D.P. 3		
SET GROUP 1		
Σ_n 0	0.000	CL
2 1 . 7 Σ_{nxx^2}	21.700	Σ 1
2 8 . 5 Σ_{nxx^2}	28.500	Σ 1
3 1 . 2 Σ_{nxx^2}	31.200	Σ 1
2 4 . 6 Σ_{nxx^2}	24.600	Σ 1

2 7 . 0 Σ_{nxx^2}	27.000	Σ 1
3 8 . 1 Σ_{nxx^2}	38.100	Σ 1
2 3 . 2 Σ_{nxx^2}	23.200	Σ 1
SD MEAN	5.588	SDn-1 1
2ND FUNC	27.757	F 2
Φ_n 3	3.223	F 3

B. Compute the Z-statistic and normal probability associated with each score.

2 1 . 7 Φ_n 1	21.700	X
	-1.170	Z 1
Φ_n 2	-1.170	P z
	0.120	*
2 8 . 5 Φ_n 1	28.500	X
	0.143	Z 1
Φ_n 2	0.143	P z
	0.557	*
3 1 . 2 Φ_n 1	31.200	X
	0.665	Z 1
Φ_n 2	0.665	P z
	0.747	*

2 4 . 6 Φ_n 1	24.600 X
	-0.610 Z 1
Φ_n 2	-0.610 P z
	0.270 *
2 7 Φ_n 1	27.000 X
	-0.146 Z 1
Φ_n 2	-0.146 P z
	0.441 *
3 8 . 1 Φ_n 1	38.100 X
	1.999 Z 1
Φ_n 2	1.999 P
	0.977 *
2 3 . 2 Φ_n 1	23.200 X
	-0.880 Z 1
Φ_n 2	-0.880 P z
	0.189 *

C. Delete the 38.1 and add seven 26.1's to the group.

3 8 . 1 DELETE Σ_{nxx^2}	38.100 Σ -1
2 6 . 1 Σ_{nxx^2}	26.100 Σ 1
7 =	7.000 n

D. Compute the new standard deviation, mean and standard error of the mean for the revised group.

SD MEAN	2.283 SDn-1 1
2ND FUNC	26.069 F 2
Φ_n 3	0.633 F 3

E. Compute the z-statistic and normal probability associated with each score in the new group.

2 1 . 7 Φ_n 1	21.700 X -1.991 Z 1
Φ_n 2	-1.991 P z 0.023 *
2 8 . 5 Φ_n 1	28.500 X 1.107 Z 1
Φ_n 2	1.107 P z 0.866 *
3 1 . 2 Φ_n 1	31.200 X 2.338 Z 1
Φ_n 2	2.338 P z 0.990 *

2 4 . 6 Σ+ 1	24.600 X
	-0.669 Z 1
Σ+ 2	-0.669 P z
	0.251 *
2 7 Σ+ 1	27.000 X
	0.424 Z 1
Σ+ 2	0.424 P z
	0.664 *
2 3 . 2 Σ+ 1	23.200 X
	-1.307 Z 1
Σ+ 2	-1.307 P z
	0.095 *
2 6 . 1 Σ+ 1	26.100 X
	0.014 Z 1
Σ+ 2	0.014 P z
	0.505 *

Problem 2

Here's an example of linear regression using transformed data. A two-variable regression is used in the example, but three-variable may be done the same way.

A. Do a linear regression using these raw data:

X	Y
1.0	1.5
3.5	3.5
5.5	4.0
6.0	6.5
6.5	9.0
5.5	11.0
7.0	11.5

[SET D. P.] [3]		
[Φn] [0]	0.000	CL
[1] [XY]	1.000	X
[1] [.] [5] [=]	1.500	Y
[3] [.] [5] [XY]	3.500	X
[3] [.] [5] [=]	3.500	Y
[5] [.] [5] [XY]	5.500	X
[4] [=]	4.000	Y

6 XY	6.000	X
6 . 5 =	6.500	Y
6 . 5 XY	6.500	X
9 =	9.000	Y
5 . 5 XY	5.500	X
1 1 =	11.000	Y
7 XY	7.000	X
1 1 . 5 =	11.500	Y
SET GROUP 1		
LIN REG	0.805	LR 1
2ND FUNC	1.509	F 2
Φ_n 3	-0.833	F 3

Therefore, r = .805 and Y_{est} = 1.509X - 0.833.

B. Now let's do a regression on these data using Ln(Y) in place of Y.

[Σn] [0]	0.000	CL
[1] [XY]	1.000	X
[1] [.] [5] [Ln LOG]	1.500 0.405	lg *
[=]	0.405	Y
[3] [.] [5] [XY]	3.500	X
[3] [.] [5] [Ln LOG]	3.500 1.252	lg *
[=]	1.252	Y
[5] [.] [5] [XY]	5.500	X
[4] [Ln LOG]	4.000 1.386	lg *
[=]	1.386	Y
[6] [XY]	6.000	X
[6] [.] [5] [Ln LOG]	6.500 1.871	lg *
[=]	1.871	Y

6 . 5 XY	6.500 X
9 Ln LOG	9.000 lg
	2.197 *
=	2.197 Y
5 . 5 XY	5.500 X
1 1 Ln LOG	11.000 lg
	2.397 *
=	2.397 Y
7 XY	7.000 X
1 1 . 5 Ln LOG	11.500 lg
	2.442 *
=	2.442 Y
SET GROUP 1	
LIN REG	0.913 LR 1
2ND FUNC	0.324 F 2
Σn 3	0.083 F 3

Here, $r = .913$ and $LnY_{est} = .324X + .083$.

$Y_{est} = \exp\ (.324X + .083)$

You can use the normal functions on the 445 to transform any variable during XY data entry. If you want to use an algebraic expression to transform a variable, enclose the expression in parentheses, like this:

enter X, [XY]

[(]

enter Y

[+] [5] [.] [5] [÷] [3] [)] [XY]

enter Z, [=]

This series of steps has entered X, $\frac{Y + 5.5}{3}$ and Z.

C. You can also do a quadratic regression, such as

$$Z = i + m_1X + m_2X^2$$

You do it like this:

enter X, [XY]

[(] [X] [)] [XY]

enter Z, [=]

Problem -- perform a quadratic regression on these data:

X	Z
2.0	3.1
3.1	4.9
3.8	6.2
3.6	5.6

[Σn] [0]	0.000	CL
[2] [XY]	2.000	X
[(]	2.000	(
[X]	2.000	X
[)]	2.000	)
	4.000	*
[XY]	4.000	Y
[3] [.] [1] [=]	3.100	Z
[3] [.] [1] [XY]	3.100	X
[(]	3.100	(
[X]	3.100	X

)	3.100	)
	9.610	*
XY	9.610	Y
4 . 9 =	4.900	Z
3 . 8 XY	3.800	X
(	3.800	(
X	3.800	X
)	3.800	)
	14.440	*
XY	14.440	Y
6 . 2 =	6.200	Z
3 . 6 XY	3.600	X
(	3.600	(
X	3.600	X
)	3.600	)
	12.960	*
XY	12.960	Y
5 . 6 =	5.600	Z

SET GROUP 4

LIN REG

2ND FUNC

Φ_n 3

0.762	LR	4
0.909	F	2
0.131	F	3

Therefore, the regression equation for these data is

$$Z_{est} = .762 + .909X + .131X^2 .$$

Problem 3

Here we'll be concerned with multiple and partial correlation coefficients for three-variable linear regression. From the three two-variable correlation coefficients that can be calculated between the pairs XY, XZ and YZ, both the multiple and partial correlation coefficients can be calculated. For simplicity, in this example we define X as Group 1, Y as Group 2 and Z as Group 3. Thus, the correlation coefficients calculated by the two-variable linear regression are r_{12}, r_{23} and r_{31}.

First, let's put in some three-variable data.

X	Y	Z
1	2	3
7	8	9
1.5	8.6	3.1
7.5	4.6	1.8

[Φη] [0]	0.000	CL
[1] [XY]	1.000	X
[2] [XY]	2.000	Y
[3] [=]	3.000	Z
[7] [XY]	7.000	X
[8] [XY]	8.000	Y
[9] [=]	9.000	Z
[1] [.] [5] [XY]	1.500	X
[8] [.] [6] [XY]	8.600	Y
[3] [.] [1] [=]	3.100	Z

7 . 5 XY	7.500 X
4 . 6 XY	4.600 Y
1 . 8 =	1.800 Z

A. Calculate the multiple correlation coefficient of X on Y and Z as given by

$$R_{1.23} = \sqrt{\frac{r_{12}^2 + r_{31}^2 - 2r_{12}r_{23}r_{31}}{1 - r_{23}^2}}$$

Here's one good way to do it:

RESET	0.000 ^
(	0.000 (
SET GROUP 1 LIN REG	0.211 LR 1
X	0.211 X
)	0.211)
	0.044 *
+	0.044 +

(	0.044 (
SET GROUP 3 LIN REG	0.365 LR 3
X	0.365 X
)	0.365) 0.133 *
-	0.133 -
(	0.177 (
2 X	2.000 X
SET GROUP 1 LIN REG	0.211 LR 1
X	0.211 X
SET GROUP 2 LIN REG	0.497 LR 2
X	0.497 X
SET GROUP 3 LIN REG	0.365 LR 3
)	0.365) 0.076 *
÷	0.076 ÷

(	0.101 (
1 -	1.000 -
(	1.000 (
SET GROUP 2 LIN REG	0.497 LR 2
X	0.497 X
)	0.497) 0.247 *
)	0.247) 0.752 *
=	0.752 = 0.134 *
√	0.134 √ 0.366 *
	0.366

B. Calculate the multiple correlation coefficient of Y on X and Z as given by

$$R_{2.13} = \sqrt{\frac{r_{12}^2 + r_{23}^2 - 2r_{12}r_{23}r_{31}}{1 - r_{31}^2}}$$

This expression can be evaluated in this way:

$$(r_{12}^2) + (r_{23})^2 - (2 \times r_{12} \times r_{23} \times r_{31}) \div (1 - (r_{31}^2))$$

then [=] [√] . The result is $R_{2.13} = .498$. Here's a tape which shows one way of doing it.

0.000	^	
0.000	(	
0.211	LR	1
0.211	X	
0.211	)	
0.044	*	
0.044	+	
0.044	(	
0.497	LR	2
0.497	X	
0.497	)	
0.247	*	
0.247	-	
0.292	(	
2.000	X	
0.211	LR	1
0.211	X	
0.497	LR	2
0.497	X	
0.365	LR	3
0.365	)	

0.076	*
0.076	÷
0.215	(
1.000	-
1.000	(
0.365	LR 3
0.365	X
0.365	)
0.133	*
0.133	)
0.866	*
0.866	=
0.248	*
0.248	√
0.498	*
0.498	

C. Now calculate the multiple correlation coefficient of Z on X and Y as given by

$$R_{3.12} = \sqrt{\frac{r_{31}^2 + r_{23}^2 - 2r_{12}r_{23}r_{31}}{1 - r_{12}^2}}$$

This expression can be evaluated in this way:

$$(r_{31}^2) + (r_{23})^2 - (2 \times r_{12} \times r_{23} \times r_{31}) \div (1 - (r_{12}^2))$$

then [=] [√‾] . The result is $R_{3.12} = .564$.

D. Calculate the partial correlation coefficient between X and Y as given by

$$r_{12.3} = \frac{r_{12} - r_{31}r_{23}}{\sqrt{(1 - r_{31}^2)\ (1 - r_{23}^2)}}$$

Here's one way of doing it:

[RESET]	0.0000 ^
[(]	0.0000 (
[1] [-]	1.0000 -
[(]	1.0000 (
[SET GROUP] [3] [LIN REG]	0.3650 LR 3
[X]	0.3650 X
[)]	0.3650)
	0.1332 *
[)]	0.1332)
	0.8667 *
[X]	0.8667 X
[(]	0.8667 (

1 -	1.0000 -
(	1.0000 (
SET GROUP 2 LIN REG	0.4978 LR 2
X	0.4978 X
)	0.4978) 0.2478 *
)	0.2478) 0.7521 *
=	0.7521 = 0.6519 *
√	0.6519 √ 0.8074 *
1/x	0.8074 1/x 1.2385 *
X	1.2385 X
(	1.2385 (
SET GROUP 1 LIN REG	0.2113 LR 1
-	0.2113 -
(	0.2113 (

SET GROUP 3 LIN REG	0.3650 LR 3
X	0.3650 X
SET GROUP 2 LIN REG	0.4978 LR 2
)	0.4978)
	0.1817 *
)	0.1817)
	0.0296 *
=	0.0296 =
	0.0366 *
	0.0366

As you can see on the tape, $r_{12.3} = .0366$.

E. Calculate the partial correlation coefficient between Y and Z as given by

$$r_{23.1} = \frac{r_{23} - r_{12}r_{31}}{\sqrt{(1 - r_{12}^2)\ (1 - r_{31}^2)}}$$

This expression can be evaluated in this way:

$(1 - (r_{12}^2)\ ) \times (1 - (r_{31}^2)\ )$

then [=] [$\sqrt{}$] [$\frac{1}{x}$] [÷] $(r_{23} - (r_{12} \times r_{31})\ )$ and [=] .

The result is $r_{23.1} = .4623$. Here's a tape which shows one way to do it:

```
0.0000      ^
0.0000      (
1.0000      -
1.0000      (
0.2113      LR    1
0.2113      X
0.2113      )
0.0446      *
0.0446      )
0.9553      *
0.9553      X
0.9553      (
1.0000      -
1.0000      (
0.3650      LR    3
0.3650      X
0.3650      )
0.1332      *
0.1332      )
0.8667      *
0.8667      =
```

0.8280	*
0.8280	√
0.9099	*
0.9099	1/x
1.0989	*
1.0989	X
1.0989	(
0.4978	LR 2
0.4978	-
0.4978	(
0.2113	LR 1
0.2113	X
0.3650	LR 3
0.3650	)
0.0771	*
0.0771	)
0.4207	*
0.4207	=
0.4623	*
0.4623	

F. Calculate the partial correlation coefficient between Z and X as given by

$$r_{31.2} = \frac{r_{31} - r_{12}r_{23}}{\sqrt{(1 - r_{12}^2)\ (1 - r_{23}^2)}}$$

The result is $r_{31.2} = .3065$.

Problem 4

Fisher's exact test gives a probability statement for a 2X2 matrix,

A	B	n_1
C	D	n_2
n_a	n_b	N

$n_1 = A + B$
$n_2 = C + D$
$n_a = A + C$
$n_b = B + D$
$N = A + B + C + D$

$$p = \frac{n_1 ! n_2 ! n_a ! n_b !}{N! A! B! C! D!}$$

Calculate p for the following matrix:

2	5	7
7	4	11
9	9	18

Here, $p = \frac{7!11!9!9!}{18!2!5!7!4!}$.

Here's how you do it on the machine:

[7] [Φn] [6]	7.000 !
	5,040.000 *
[X]	5,040.000 X
[1] [1] [Φn] [6]	11.000 !
	39,916,800.00 *
[X]	39,916,800.00 X
[9] [Φn] [6]	9.000 !
	362,880.000 *
[X]	362,880.000 X
[9] [Φn] [6]	9.000 !
	362,880.000 *
[÷]	362,880.000 ÷
[(]	2.649185200 22 (
[1] [8] [Φn] [6]	18.000 !
	6.402373705 15 *
[X]	6.402373705 15 X
[2] [Φn] [6]	2.000 !
	2.000 *
[X]	2.000 X
[5] [Φn] [6]	5.000 !
	120.000 *
[X]	120.000 X

7	
Φ_n	
6	
X	
4	
Φ_n	
6	
)	
=	

7.000	!
5,040.000	*
5,040.000	X
4.000	!
24.000	*
24.000	)
1.858634696	23 *
1.858634696	23 =
0.142	*

Problem 5

Here, we shall take several different kinds of means for the following data:

21.7

28.5

31.2

24.6

27.0

23.2

(7x) 26.1

A. Calculate the Harmonic Mean for the above data.

$$\text{Harmonic Mean} = \frac{N}{\sum \frac{1}{X}}$$

SET GROUP 2 ±n 0	0.000	CL
2 1 . 7 1/x	21.700	1/x
	0.046	*
Σ nxx2	0.046	Σ 2
2 8 . 5 1/x	28.500	1/x
	0.035	*
Σ nxx2	0.035	Σ 2
3 1 . 2 1/x	31.200	1/x
	0.032	*
Σ nxx2	0.032	Σ 2
2 4 . 6 1/x	24.600	1/x
	0.040	*
Σ nxx2	0.040	Σ 2
2 7 1/x	27.000	1/x
	0.037	*
Σ nxx2	0.037	Σ 2
2 3 . 2 1/x	23.200	1/x
	0.043	*
Σ nxx2	0.043	Σ 2
2 6 . 1 1/x	26.100	1/x
	0.038	*

Σ_{nxx^2}	0.038 Σ 2
7 =	7.000 n
RCL$_n$ 4	13.000 ↑ 4
÷	13.000 ÷
RCL$_n$ 5	0.502 ↑ 5
=	0.502 = 25.885 *

The Harmonic Mean is 25.885.

B. Take the Root Mean Square for the same data as above.

$$rms = \sqrt{\frac{\Sigma X^2}{N}}$$

SET GROUP 3 Φ↓ 0	0.000 CL
2 1 . 7 Σ_{nxx^2}	21.700 Σ 3
2 8 . 5 Σ_{nxx^2}	28.500 Σ 3
3 1 . 2 Σ_{nxx^2}	31.200 Σ 3

24 . 6 Σ_{nxx^2}	24.600	Σ 3
27 Σ_{nxx^2}	27.000	Σ 3
23 . 2 Σ_{nxx^2}	23.200	Σ 3
26 . 1 Σ_{nxx^2}	26.100	Σ 3
7 =	7.000	n
RCL_n 9	8,897.450	↑ 9
÷	8,897.450	÷
RCL_n 7	13.000	↑ 7
=	13.000	=
	684.419	*
√	684.419	√
	26.161	*

The Root Mean Square is 26.161.

C. Now let's take a Geometric Mean for these data.

$$\text{Geometric Mean} = \sqrt[N]{(X_1)\,(X_2)\,(X_3)\ldots(X_n)}$$

RESET	0.000	^
21.7 X	21.700	X
28.5 X	28.500	X
31.2 X	31.200	X
24.6 X	24.600	X
27 X	27.000	X
23.2 X	23.200	X
(	297,335,006.8	(
26.1 a^x	26.100	a^x
7)	7.000	)
	8,250,562,363.	*
a^x	8,250,562,363.	a^x
13 $\frac{1}{x}$	13.000	1/x
	0.076	*
=	0.076	=
	25.977	*

The Geometric Mean is 25.977.

PROGRAMMING EXAMPLES

Try Stirling's Formula for log (n !). Here, n is in register 4 and has the value 1728. (Before you start, put 1728 into register 4.)

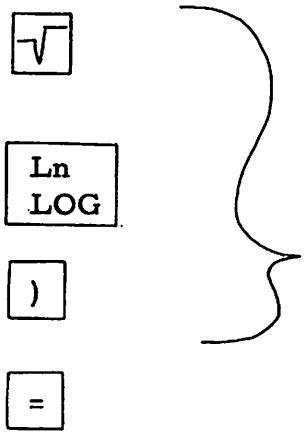
$$\text{Ln}(n!) \cong (n+\tfrac{1}{2})\ \text{Ln}(n) - n + \text{Ln}\sqrt{2\pi}$$

RCL_n 4	1,728.000	↑ 4
+	1,728.000	+
. 5 X	0.500	X
RCL_n 4	1,728.000	↑ 4
Ln LOG	1,728.000 7.454	lg *
-	7.454	-
RCL_n 4	1,728.000	↑ 4
+	1,728.000	+
(	11,157.483	(
π e +	3.141	+
=	3.141 6.283	= *

$(n+\frac{1}{2})\log n$

doubles π

$\log\sqrt{2\pi}$



log $\sqrt{2\pi}$
(continued)

```
      6.283      √
      2.506          *

      2.506        lg
      0.918          *

      0.918              )


      0.918            =
 11,158.402            *
```

Let's program Stirling's Formula. We'll put it at location 0030, give it the symbolic name "FLAG" and assume n is in register 4.

Do the following:

RESET just a precaution - not usually necessary

BRANCH nn (or could be JUMP nn) 0 3

put switch on LOAD

Now you're loading a program - the program will be the same sequence of keys as before, but with a SYMBOL in the front and PRINT and HALT instructions at the end.

IND SYMB

FLAG

```
      0.000          ^

0030      066


0031      016
```

RCL_n	0032	111	↑
4	0033	004	4
+	0034	021	+
.	0035	012	
5	0036	005	5
X	0037	023	X
RCL_n	0038	111	↑
4	0039	004	4
Ln LOG	0040	050	lg
-	0041	022	-
RCL_n	0042	111	
4	0043	004	4
+	0044	021	+
(	0045	026	(
π e	0046	015	

+	0047	021	+
=	0048	020	=
√	0049	055	√
Ln LOG	0050	050	lg
)	0051	027	)
=	0052	020	=
PRINT ANS	0053	061	A
HALT	0054	056	
JUMP nn	0055	126	Ju
IND SYMB	0056	067	
FLAG	0057	016	

Now take the switch off LOAD and put it on RUN, before you forget it's in LOAD and accidentally load in another step.

Put 1728 in register 4 (1728 ST_n **4** **) and push** **RESUME** **.**

If you didn't turn off the PRINT switch, all the steps will print out -- so turn it off. The answer will print, and the program will stop. Now you can store another number in register 4, press RESUME and the 445 will do it all over again.

Now try a program with some logic in it. The quadratic is a good one.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

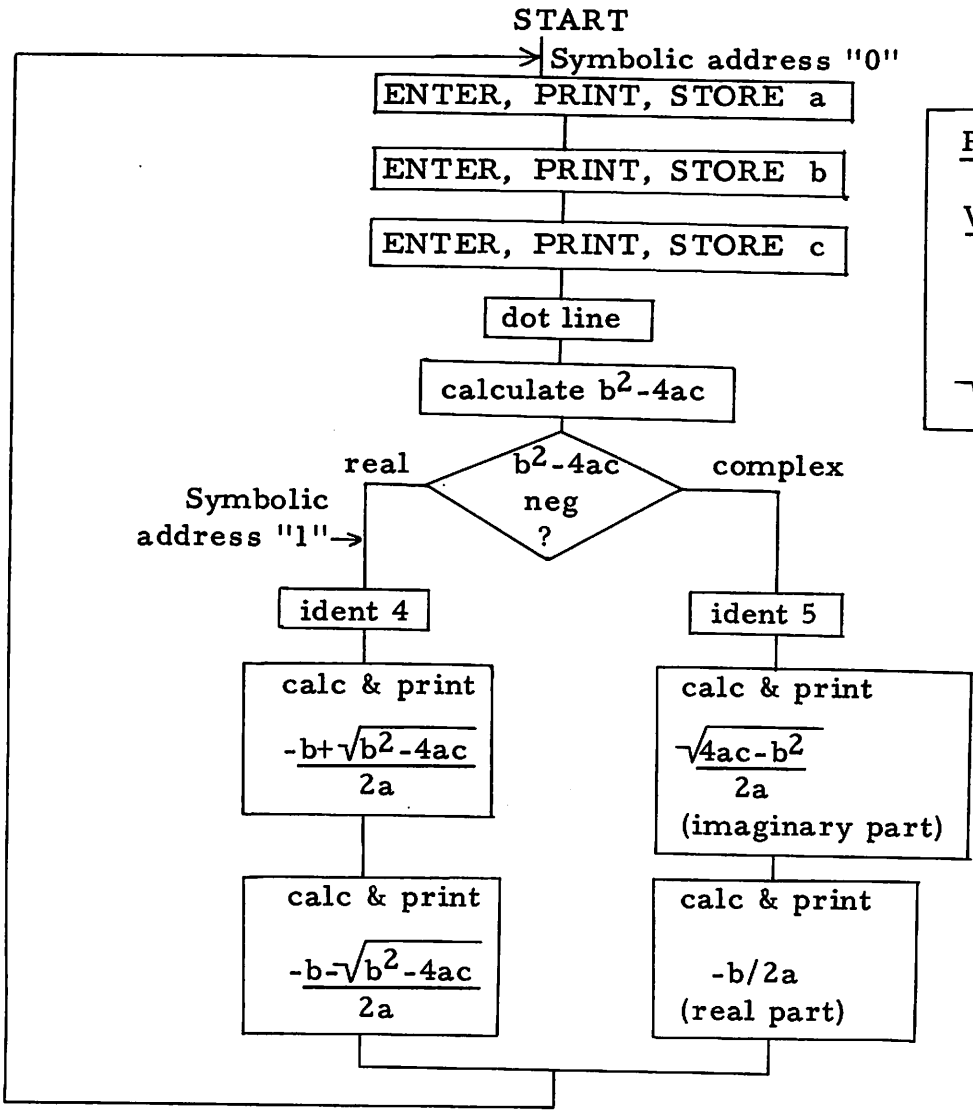
where $ax^2 + bx + c = 0$

Now we'll use identifiers and dot lines to indicate what's going on. We want the tape to print identifiers 1, 2 and 3 to indicate a, b, and c; and identifiers 4 and 5 to indicate whether the roots are real or imaginary. So a sample tape might look like this for a = 17, b = 18, c = 19:

```
1.
                  17.00
2.
                  18.00
3.
                  19.00
.....................

5.
                   0.91
                  -0.52
```

Here's a flow chart for the program:



Register Usage Chart

Variable	Register
a	0
b	1
c	2
$\sqrt{b^2-4ac}$	3

Here's the program. Since it's written with symbolic addressing, it can go anywhere in memory. It happens to start at step 0140.

```
SYMB
0                    Symbolic address "0"
1
IDENT                identifier 1
HALT
PRINT ENTRY
ST_n                 enter, print &
0                    store a
2
```

```
0140    066
0141    000          0
0142    001          1
0143    177
0144    056
0145    060
0146    110          ↓
0147    000          0
0148    002          2
```

IDENT	identifier 2	0149	177	
HALT		0150	056	
PRINT ENTRY	enter, print &	0151	060	
ST_n	store b	0152	110	↓
1		0153	001	1
3		0154	003	3
IDENT	identifier 3	0155	177	
HALT		0156	056	
PRINT ENTRY	enter, print &	0157	060	
ST_n	store c	0158	110	↓
2		0159	002	2
DOT PRINT	dot line	0160	176	
RCL_n		0161	111	↑
1		0162	001	1
X		0163	023	X
-		0164	022	-
(		0165	026	(
4		0166	004	4
X		0167	023	X
RCL_n	b^2-4ac	0168	111	↑
0		0169	000	0
X		0170	023	X
RCL_n		0171	111	↑
2		0172	002	2
)		0173	027	)
=		0174	020	=
JUnn	Jump	0175	126	Ju
+	if positive	0176	021	+
SYMB	to symbolic	0177	067	
1	address "1"	0178	001	1
5	otherwise,	0179	005	5
IDENT	ident 5, - complex roots	0180	177	
CHG SIGN	$4ac-b^2$ in entry register	0181	013	-
$\sqrt{\ }$		0182	055	$\sqrt{\ }$
÷		0183	024	÷
2		0184	002	2
÷	$\frac{\sqrt{4ac-b^2}}{2a}$	0185	024	÷
RCL_n		0186	111	↑
0		0187	000	0
=		0188	020	=
PRINT ENTRY	Print imag. term	0189	060	

RCL_n		
1		
CHG SIGN		
÷		
2	-b/2a	
÷		
RCL_n		
0		
=		
PRINT ENTRY	Print real term	
ADV		
JUnn		
SYMB		
0	Return to start	
SYMB		
1	Begin real root routine	
4		
IDENT	Ident 4 - real roots	
$\sqrt{\ }$		
ST_n	Store $\sqrt{b^2-4ac}$ for later use	
3		
-		
RCL_n		
1	$\frac{-b+\sqrt{b^2-4ac}}{2a}$	
÷		
2		
÷		
RCL_n		
0		
=		
PRINT ENTRY	1st real root	
RCL_n		
3		
CHG SIGN	Calculate	
-		
RCL_n	$\frac{-b-\sqrt{b^2-4ac}}{2a}$	
1		
÷		
2		
÷		
RCL_n		
0		
=		
PRINT ENTRY	2nd real root	
ADV		
JUnn		
SYMB		
0	Back to START	

0190	111	↑
0191	001	1
0192	013	-
0193	024	÷
0194	002	2
0195	024	÷
0196	111	↑
0197	000	0
0198	020	=
0199	060	
0200	065	
0201	126	Ju
0202	067	
0203	000	0
0204	066	
0205	001	1
0206	004	4
0207	177	
0208	055	$\sqrt{\ }$
0209	110	↓
0210	003	3
0211	022	-
0212	111	↑
0213	001	1
0214	024	÷
0215	002	2
0216	024	÷
0217	111	↑
0218	000	0
0219	020	=
0220	060	
0221	111	↑
0222	003	3
0223	013	-
0224	022	-
0225	111	↑
0226	001	1
0227	024	÷
0228	002	2
0229	024	÷
0230	111	↑
0231	000	0
0232	020	=
0233	060	
0234	065	
0235	126	Ju
0236	067	
0237	000	0

After you've entered the program, put the switch back to RUN and push RESUME to get the first identifier printed out. Then enter the sample problem (a=17, b=18, c=19). Your tape should look like the sample above.

CHAPTER SIX

MAGNETIC CARDS AND OTHER IMPORTANT THINGS

MAGNETIC CARDS

That little slot above the keyboard is for mag (magnetic) cards. You can record either a program or data on these cards and store them for ever and ever. Then, whenever you need that program or data again, just stick in the card and it's in the machine. Automatically. Each card has two edges, and each edge can hold up to 256 program steps or 32 data registers. So each card can hold up to 512 program steps or 64 data registers. The ends (with the oblong holes) are the ones you stick into the machine, and you can write with a felt-tip pen on a half-inch strip lengthwise down the middle of the light side of the card to identify what's on each edge and which end to stick in first. You can protect the contents of a card permanently by cutting off the thin strip of card material on the outer edge alongside the oblong hole. Then the oblong hole becomes somewhat of a rectangle along the leading edge of the card. Once you've done that, you can never record anything else on that card again — not even by mistake. But the information on the card can be read just the same.

Recording A Program

Access the first instruction to be written with JUMP, BRANCH or symbolic addressing, put the mag card switch on RECORD and stick the mag card into the slot -- firmly, but gently -- with the dark side

down. The card reader/writer unit will grab the card, pull it in, fill it with 256 program steps and then push it out again. If you want to record more or less than 256 steps, use the keyboard numeral keys to enter the total number of steps you're going to record just before you start sticking in cards (and after you've branched, jumped or symbolically addressed to the first location.) Then stick in a card and wait for it to come out. If you've asked for less than 256 steps, that's it. If you've asked for more, turn the card around, stick it in again, stick in a second card, etc. When you stick in the first card, the IDLE light will go off. And it will stay off until you've stuck in enough cards to record the number of steps you've asked for.

If the program instructions you want to record don't start at a branch point or symbolically addressed location, you have to use a very specific series of steps to record them on mag cards. And you have to do it in this order:

1) Put the mag card switch on RECORD.
2) JUMP or BRANCH to a nearby branch point.
3) If you will be recording more or less than 256 steps, enter the number of steps now.
4) Put the switch on LOAD.
5) LIST or BACK SPACE to the instruction before the first step you want recorded on the card.

6) Stick in the required number of cards.

7) Don't forget to take the machine out of LOAD.

Recording Data

The procedure for recording data (registers) on mag cards is essentially the same as recording a program, with the following exceptions:

1) Access the first register (the beginning of your data series) with [ST nn] or [RCL nn] instead of JUMP, BRANCH, etc.

2) The number of information units that can be recorded on each edge is 32 registers (instead of 256 steps). If you're going to record more or less than 32 registers, enter the number of registers you need just before sticking in the first card.

Entering A Program

Access the address in program memory where you wish the program to begin. Then put the mag card switch on ENTER. Now just start sticking in cards (in their proper order, of course). The machine will load all program steps, up to the capacity of program memory, in sequential order as they are read, no matter how many cards have to be fed in. If your card only contains 25 steps, however, the machine will enter only the 25 steps and then spit the card out.

Entering Data

Access the register where you want the data to begin. Put the mag

switch on ENTER. Then start sticking in cards until all the data are entered. The machine will load data sequentially into consecutive registers until there is no more information being fed in. If there are less than 32 registers on your card, however, the machine will enter what's there and then spit the card out.

Verification Of Card Information

Each time information is recorded on a mag card, a special number — consisting of the sum of all the numbers contained on the card — is also recorded. Then, whenever the card is entered, the machine adds up this sum again and compares it with the number recorded on the card. If they're not the same the machine goes into ERROR mode.

Care Of Magnetic Cards

Mag cards are pretty durable. But they still require proper care. Write on them only with a felt-tip pen. And write in the half-inch strip running lengthwise down the middle of the light side. The dark side is the magnetic side, and should be kept from contamination (coffee, glue, tears, etc.). Don't store mag cards with their magnetic surfaces against each other. Use the envelopes provided. And try not to bend them too much. If you take care of them, these cards can last virtually forever.

ACCURACY OF THE 445

As we've mentioned, the 445 holds a 13-digit signed mantissa with a 2-digit signed exponent for all numbers. This is true of the data registers as well as the entry register. In addition there are some specific levels of accuracy to consider. And these are them:

Function	Accuracy
Add	13 digits
Subtract	13 digits
Multiply	13 digits, rounded to the 13th digit
Divide	13 digits, rounded to the 13th digit (i.e., $2 \div 3 = 0.6666666666667$)
a^x	± 1 in the 11th digit
Invert $(\frac{1}{x})$	Same as divide
Square root	± 3 in the 13th digit
Logarithm	± 1 in the 11th digit
Antilogarithm	± 1 in the 11th digit
Standard deviation	± 5 in the 13th digit
Normal probability	to 6 fractional digits
z-statistic	± 5 in the 13th digit
X^2 statistic	± 5 in the 13th digit
Linear regression	± 5 in the 13th digit

PERIPHERAL EQUIPMENT

In addition to all the things the 445 will do by itself, there are some specialized things it can do with the addition of peripheral equipment. Listed below are the peripheral devices designed for use with the 445 Statistician.

Model 490 - Mark Sense/Punch Card Reader

This device will read program steps and data from punched paper cards or mark sense cards and store the information directly into the 445 memory.

Model 492 - Magnetic Tape Cassette Unit

Reads and writes data or program steps (or a combination of both) on standard digital cassettes. You can store up to 100,000 program steps or 12,500 data registers on a single cassette.

Model 493 - X Y Plotter

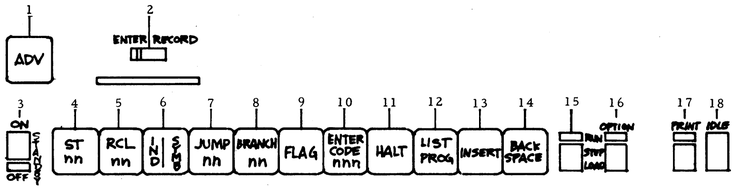
This unit transforms computational results into graphic form. It plots functions, makes graphs, draws axes and does alphanumeric labelling.

Model 494 - Typewriter

There are two versions of the typewriter:

1) Output Only Typewriter - Does automatic form preparation. It can, for instance, prepare formatted, tabular documents with alphanumeric headings and complete tab facilities.
2) Input/Output Typewriter - Has the same output capability as above, with the additional capacity to accept alphanumeric input from the typewriter keyboard.

KEY FUNCTION GLOSSARY



1. ADVANCE advances the tape. If you press it and release it, the tape will advance one space. If you hold it down, the tape will continue advancing until you let up.

2. The ENTER/RECORD switch controls the magnetic card read/write unit. On ENTER, the unit will read information off the card and enter it into the machine. On RECORD, it will write information onto the card from the machine.

3. The ON/STANDBY/OFF switch controls the degree of "offness" of the machine. On ON, everything works normally. On STANDBY, no operations can be initiated, but all data and program information that have been fed into the 445's memory will remain intact. On OFF, all operational capacity is shut off and all information in all memories will be lost. Switching the machine from OFF to ON clears all registers, fills program memory with NOOP codes, sets the decimal point to two places, sets Group 1 and executes RESET.

4. STORE nn allows you to store information in the main data storage registers. When used in conjunction with the arithmetic keys, the STORE nn key also allows you to perform arithmetic operations in those registers.

5. RECALL nn allows you to recall information from the main data storage registers. When used in conjunction with the arithmetic keys, RECALL nn also allows you to perform arithmetic operations out of those registers and into the entry register.

6. INDIRECT/SYMBOL performs two functions. It can be used in conjunction with BRANCH and JUMP instructions for symbolic addressing of program steps. And it may be used with the main data STORE, RECALL and EXCHANGE instructions for indirect addressing of main data registers.

7. JUMP nn lets you jump to a specific location in program memory.

8. BRANCH nn lets you branch directly to a specific location in program memory, like the JUMP nn key, but also records the address from which you branched. You can return to the address directly following the location from which you branched with the RESUME instruction.

9. FLAG is a simple condition you can use in a program. This key sets Flag 1. Having your program test for Flag 1 is a convenient device for conditional branching or jumping.

10. ENTER CODE nnn allows you to enter a function code directly into your program.

11. HALT enters a halt instruction into your program, causing the program to stop at that point for data entry or other operation. It also causes the cessation of program running or listing.

12. LIST PROGRAM causes the address, code and print symbol of a location in your program to print. It may be used to list the entire program (in LOAD mode), or to identify your current location (in RUN mode). Your program steps will be listed, but not executed.

13. INSERT lets you add a step to any part of your program.

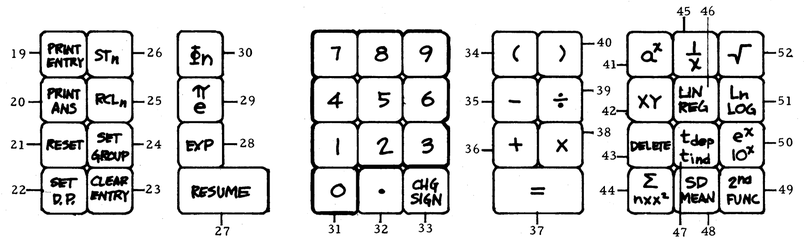
14. BACK SPACE acts exactly like the back space on a typewriter when the 445 is in LOAD mode, causing the machine to back up one space in program memory.

15. The RUN-STEP-LOAD switch controls the operation mode of the machine. In RUN, either keyboard or programmed operations may be performed. In LOAD, a program may be entered into program memory. In STEP, you can go through your program one step at a time - each time you press the RESUME key, the next step in your program will be executed.

16. The OPTION switch is another simple condition which your program can test. But OPTION may be changed manually at any time, allowing you to control physically the results of a test using the OPTION device.

17. The PRINT switch can be UP or DOWN. When it's UP the 445 will print nearly everything it does. When it's DOWN, it will print only a few "print always" functions.

18. The IDLE light indicates when the 445 is busy. It is off whenever the machine is performing an operation, and will go on again when it's ready for more instructions. (Two additional keys may be pressed even if the IDLE light is off.) The IDLE light will flash when the 445 is in OVERFLOW or ERROR mode.



19. PRINT ENTRY causes the contents of the entry register to be printed with the number of places indicated by the decimal point setting, but without rounding. Digits beyond the number of places called for will simply be ignored.

20. PRINT ANSWER causes the contents of the entry register to be rounded off to the number of places indicated by the decimal point setting, and the rounded answer to be printed.

21. RESET clears the entry register, the List-Mode Accumulator and an ERROR or OVERFLOW condition. When pushed during a multiple-key sequence, it will interrupt the operation in progress and reset everything back to zero.

22. SET DECIMAL POINT allows you to control the number of decimal places that will print. Numerals 0-8 entered after this key will determine the number of places to the right of the decimal point. Numeral 9 indicates exponential mode.

23. CLEAR ENTRY clears only the entry register and an ERROR or OVERFLOW condition. It will not affect an operation in progress.

24. SET GROUP sets one of three data groups for summation and computation of data. It also sets Group 4, which is used for three-variable computations.

25. RECALL n will cause the contents of the scratch pad register indicated (n) to be put into the entry register. This will not affect the contents of the register recalled.

26. STORE n will cause the contents of the entry register to be put into the scratch pad register indicated (n). The value stored will replace whatever value may have been in that register before. The entry register will not be affected.

27. RESUME has two functions. It will cause a program to resume execution after a halt or after it has been loaded. And at the end of a subroutine, it will cause the program to return to the step following the instruction which caused branching to that subroutine.

28. EXPONENT enables you to enter values exponentially. You can enter up to a 13-digit signed mantissa and a two-digit signed exponent.

29. π/e gives you two values. π (13 places) will be put into the entry register and e (13 places) will be put into the second function register. To print π and leave it in the entry register, press the PRINT ENTRY key. To print e and leave it in the entry register, press the SECOND FUNCTION key.

30. Σn initiates ten additional functions available from the keyboard. Each additional function has a numeral name. (A complete list of these functions and their numeral names will be found in the strip just below the top row of keys.)

31. 0-9 are the ten numeral keys. They are used to enter numbers and the numeral names of machine functions and places.

32. This is a decimal point. It is, of course, used to enter the decimal point when you enter fractional values.

33. CHANGE SIGN changes the sign of the value in the entry register. It may be pressed any time during, before or after a numeral entry to change its sign. Or it may be pressed without numeral entry to change the sign of the value already in the entry register.

34. This key opens parentheses. Quantities may be entered as parenthetical expressions, in algebraic sequence. Up to two levels of parenthetical nesting is possible on the 445.

35. The minus sign is used to indicate subtraction. (To make a value negative, you have to use CHANGE SIGN.) Arithmetic functions are entered in their normal place in an algebraic sequence.

36. The plus sign indicates addition. It is entered in its normal place in an algebraic sequence.

37. Equals terminates an algebraic sequence and causes the result to print.

38. The times key indicates multiplication and, as with all arithmetic functions, is entered in its normal position in an algebraic sequence.

39. The divide key indicates division and is entered in its normal position in an algebraic sequence.

40. This key closes parentheses. It also completes the operation enclosed within the parentheses.

41. This key enables you to raise a number to a power. Enter the number (a). Then press this key and enter the power (x). Then press equals and the result will be printed.

42. The XY key is used for doing two- and three-variable dependent data summations.

43. The DELETE key is used for removing data from summations.

44. The sigma key is used for doing two- and three-variable independent data summations.

45. INVERT (or RECIPROCAL) takes the reciprocal of the number in the entry register, prints it, and leaves the reciprocal in the entry register.

46. This key computes the linear regression for two- and three-variable independent data. When you press the key, the correlation coefficient is printed and left in the entry register; the slope is put into the second function register and may be recalled with the SECOND FUNCTION key; and the intercept of X on the Y axis is put into the third function register and may be recalled with the THIRD FUNCTION operation.

47. This key calculates the dependent and the independent t-statistics. The dependent t is printed and left in the entry register and the independent t is put into the second function register from which it may be recalled with the SECOND FUNCTION key.

48. This key calculates the standard deviation, mean and standard error of the mean (using the n-1 method) for whatever Group is set. The standard deviation is printed and left in the entry register; the mean is put into the second function register and may be recalled with the SECOND FUNCTION key; and the standard error of the mean is put into the third function register and may be recalled with the THIRD FUNCTION operation.

49. SECOND FUNCTION exchanges the contents of the second function register with the contents of the entry register. Pressing the key a second time will return the values to their original registers.

50. This key computes the antilogarithm base-e and base-10 of the number in the entry register. The base-e antilogarithm will print and remain in the entry register. The base-10 antilogarithm will be put in the second function register and may be recalled with the SECOND FUNCTION key.

51. This key computes both the base-e and base-10 logarithms of the number in the entry register. The $\log_e$ will print and remain in the entry register. Log_{10} will be put in the second function register and may be recalled with the SECOND FUNCTION key.

52. SQUARE ROOT, as one may expect, takes the square root of the value in the entry register. The contents of the entry register will print. Then the square root will print and remain in the entry register.

APPENDIX

Functions and codes

SUMMARY OF FUNCTIONS

Following is a summary of the functions available, and their numeric codes.

Function	Page in text	On Keyboard	Numeric Code	Print Symbol
Set D. P.	17-18	Yes	117	None
Reset	17	Yes	062	∧
Clear Entry	20	Yes	063	None
Print Entry	20	Yes	060	None
Print Answer	20	Yes	061	A
Paper Advance	22	Yes	065	None
Numerals 0-9	22	Yes	000-011	None
Decimal Point	23	Yes	012	None
Exponent	23	Yes	014	None
Change Sign	23	Yes	013	None
π & e	41	Yes	015	None
Second Function	41	Yes	052	F2
Third Function	42	Yes	116, 003	F3
Plus	24	Yes	021	+
Minus	24	Yes	022	-
Multiply	24	Yes	023	X
Divide	24	Yes	024	÷
Equals	24, 38	Yes	020	=
List-Mode Add	153	No	041	+
List-Mode Subtract	153	No	042	-
List-Mode Subtotal	153	No	043	◇
List-Mode Total	153	No	040	*
a^x	25	Yes	025	a^x
Open Parenthesis	35-36	Yes	026	(
Close Parenthesis	35-36	Yes	027	)
Invert	25	Yes	054	1/x
Square Root	26	Yes	055	$\sqrt{\ }$
Logarithm	41	Yes	050	lg
Antilogarithm	41	Yes	051	lg^{-1}
Integer Fraction	43	Yes	116, 005	I
Absolute Value	154	No	045	X
Equals-Sum-Zero	155	No	037	$\bar{\Sigma}_o$
Square	154	No	053	X^2
Factorial	90	Yes	116, 006	!
Increment Entry	154	No	151	None
Decrement Entry	154	No	152	None
Identifier	157	No	177	None
Dot Print	155	No	176	None
Clear Scratch Pad Registers	46	Yes	116, 000	None
Set Group	44	Yes	115	None
Sum-Square	46	Yes	047	Σn

SUMMARY OF FUNCTIONS (Continued)

Function	Page in text	On Keyboard	Numeric Code	Print Symbol
Two Variable Data Accumulation	58	Yes	036	X, Y
Three Variable Data Accumulation	61	Yes	036	X, Y, Z
Standard Deviation (n-1)	53	Yes	077	SDn-1
Standard Deviation (n)	54	Yes	116, 004	SDn
Linear Regression	65	Yes	076	LR, n
Two Variable Linear Regression	66	Yes	076	LR, n
Three Variable Linear Regression	69	Yes	076	LR, 4
Line	72	Yes	116, 011	X, Y
Line-Two Variable	72	Yes	116, 011	X, Y
Line-Three Variable	74	Yes	116, 011	X, Y, Z
t-statistic	76	Yes	073	t
Chi-Square	85	Yes	116, 010	X^2
Delete	50	Yes	046	-
z-statistic	55	Yes	116, 001	Z
Permutation/Combination	88	Yes	116, 007	n, r, p
Normal Distribution	56	Yes	116, 002	P, z
Store Scratch Pad	26	Yes	110	↓ n
Recall Scratch Pad	26	Yes	111	↑ n
Exchange Scratch Pad	-	No	112	↕ n
Add-to-Register (Scratch Pad)	160	No	113	$+_n$ n
Total Scratch Pad	-	No	114	* n
Store Main Data	31	Yes	120	↓n n
Recall Main Data	31	Yes	121	↑n n
Exchange Main Data	155	No	122	↕n n
Add-to-Register (Main Data)	160	No	123	$+_n$ n n
Branch	128-129	Yes	127	None
Jump	128-129	Yes	126	None
Halt	99	Yes	056	None
Flag 1	152	Yes	016	None
Flag 2	154	No	017	None
Reset Flag 1	152	No	166	None
Reset Flag 2	-	No	167	None
Resume	130	Yes	057	None
Symbol/Indirect	133, 147	Yes	067	None
Print Disable	158	No	154	None
Print Enable	158	No	155	None
Program Address Keys	90, 160	Optional	105-107	None

445 FUNCTION CODES IN NUMERICAL ORDER

000	0
001	1
002	2
003	3
004	4
005	5
006	6
007	7
010	8
011	9
012	DECIMAL POINT
013	CHANGE SIGN
014	EXPONENT
015	π/e
016	Flag 1
017	Flag 2
020	= (equals)
021	+ (plus)
022	- (minus)
023	X (multiply)
024	$\div$ (divide)
025	a^x
026	((open parenthesis)
027	) (close parenthesis)
034	Permutations/combinations
035	Chi-square
036	XY data input
037	Equals-Sum-Zero
040	List-Mode Total
041	List-Mode Add
042	List-Mode Subtract
043	List-Mode Sub-total
044	Integer/Fraction
045	Absolute Value
046	DELETE
047	SUM-SQUARE
050	Ln/LOG
051	$e^x/10^x$
052	SECOND FUNCTION
053	Square
054	Invert $(\frac{1}{x})$
055	Square Root
056	HALT
057	RESUME

445 FUNCTION CODES IN NUMERICAL ORDER

060	PRINT ENTRY
061	PRINT ANSWER
062	RESET
063	CLEAR ENTRY REGISTER
065	PAPER ADVANCE
066	DEFINE SYMBOL
067	INDIRECT/SYMBOL
073	t-dependent/t-independent
075	z-statistic
076	Linear Regression
077	Standard Deviation
105	(optional)
106	(optional)
107	(optional)
110	STORE$_n$
111	RECALL$_n$
112	EXCHANGE$_n$
113	Add to scratch pad register n
114	Total scratch pad register n
115	SET GROUP
116	$\overline{\Phi}_n$
117	SET DECIMAL POINT
120	STORE$_{nn}$
121	RECALL$_{nn}$
122	EXCHANGE$_{nn}$
123	Add to register nn
126	JUMP nn
127	BRANCH nn
151	Increment entry register (+1)
152	Decrement entry register (-1)
154	PRINT DISABLE
155	PRINT ENABLE
157	RECALL DECIMAL POINT
166	Reset Flag 1
167	Reset Flag 2
176	PRINT DOT LINE
177	IDENTIFIER

INDEX

INDEX

NOTE

If the machine is turned OFF and then quickly ON again, the IDLE light may go out and the machine become inoperative. The same thing may happen if RESET or CLEAR is initiated immediately after turning the machine ON, before the automatic clearing and resetting functions are completed. To correct this condition, turn the machine OFF and leave it off for a few seconds, then turn it ON again and give it a few seconds to complete its automatic clearing and resetting operations.

Compucorp
Computer Design Corporation, 12401 West Olympic Boulevard
Los Angeles, California 90064 Telephone: (213) 478-9761