The Rockwell 900 series programmable calculators...

...Machines for people.
People for productivity.
Productivity for profit.

Programming Guide

Rockwell

ROCKWELL 900 SERIES PROGRAMMING GUIDE

INTRODUCTION

This instruction manual has been developed to aid you in learning how to program your Rockwell 900 Series Calculator.

After completion of this manual, you may write some programs which you may want to share with us. If so, we would be happy to hear from you. Please mail your programs to:

Rockwell International
950 DeGuigne Drive
Sunnyvale, California 94086

or

Sumlock Anita/Rockwell International
Anita House, Rockingham Road
Uxbridge, Middlesex, England
UB8 2XL

STANDARD LIBRARY PROGRAMS

Many of the programs that may be required could already be contained in the Rockwell Program Library Listing. Since this listing is frequently updated, an application you have that requires a program could be or will be available in the near future. In either case, contact your Rockwell salesman to see if what you want is available. Your salesman will gladly review a copy with you.

CUSTOM PROGRAMMING

There will be, of course, occasions when you want to have software written specifically for your application. If this is the case, Rockwell has initiated and is continuing to establish, a network of programmers which operate on a local, on-site basis. Our Software Department is maintaining a staff of programmers for this specific purpose. Charges for this service are nominal. Consult your Rockwell salesman for details concerning this service.

TABLE OF CONTENTS

INTRODUCTION TO FLOWCHARTING

A flowchart is a plan, or outline of the way a program should be written. It shows graphically what occurs in a program. By writing a flowchart of a program before you write the actual program steps, you will make your task much simpler. You can organize your thoughts, discover where the difficult parts of the program will be found and decide how to solve them. As a result, when you write the program, you will know exactly how to proceed.

Let's see how the flowcharting technique can be used to outline a situation that occurs every day. Consider the steps involved in selecting a new car. Assume you are going to choose between a Cadillac, a Dodge and a Volkswagen. The following flowchart will outline the steps that you would follow in deciding which car to purchase.

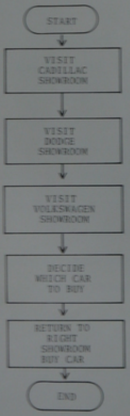


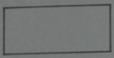
FIGURE A

Notice how this outlines the major tasks and puts them in coherent order. It is possible to re-arrange the tasks somewhat (i.e., by visiting the Dodge showroom before visiting Cadillac), but the basic logic flow cannot be changed. (You couldn't decide which car to buy before visiting at least one showroom).

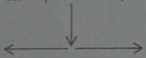
DEFINITION OF SYMBOLS

The previous flowchart used three symbols that will be applied throughout this book.

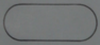
They are designed as follows:

1. 

OPERATION SYMBOL
A flowchart breaks down a program into separate operations. A description of each operation is put inside a rectangle called an "Operation Symbol."

2. 

DIRECTION ARROWS
Direction arrows show the flow directions within a flowchart.

3. 

TERMINAL SYMBOL
Terminal Symbols are used to show where a program begins and ends.

DECISION MAKING

If you study the previous flowchart, you will find that the solution as to which car to buy was not considered. You could make that decision, but how could we break down the problem so that a computer could make the decision?

First, we must establish criteria by which you would decide which car to buy. One logical decision would be based on the price of the car. Our flowchart could be designed so that it compares the price of each car with the amount of money you have to spend, and then selects the best car that you can afford.

The flowchart might look like the flowchart on the next page.

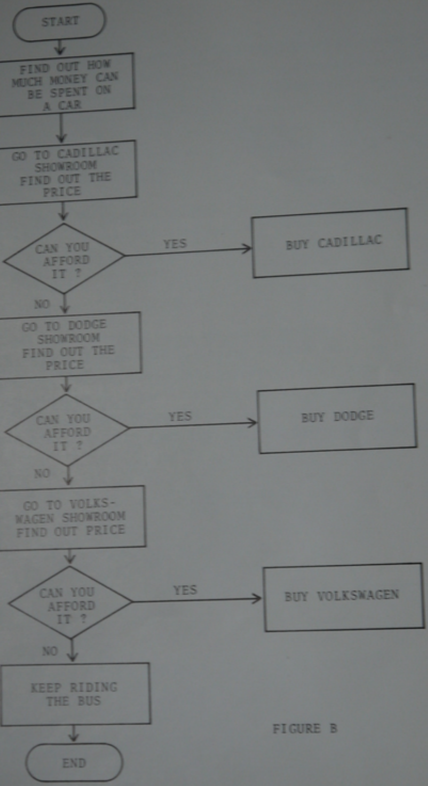
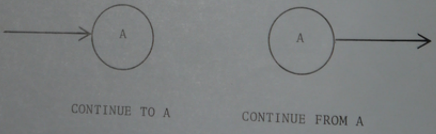

START
FIND OUT HOW MUCH MONEY CAN BE SPENT ON A CAR
GO TO CADILLAC SHOWROOM FIND OUT THE PRICE
CAN YOU AFFORD IT ?
YES
BUY CADILLAC
NO
GO TO DODGE SHOWROOM FIND OUT THE PRICE
CAN YOU AFFORD IT ?
YES
BUY DODGE
NO
GO TO VOLKS-WAGEN SHOWROOM FIND OUT PRICE
CAN YOU AFFORD IT ?
YES
BUY VOLKSWAGEN
NO
KEEP RIDING THE BUS
END

FIGURE B

The previous flowchart introduced a new symbol called a "Decision Diamond".

The Decision Diamond is used when a decision must be made. For a flowchart (or program) to work properly, each decision must be turned into a question that can be answered with yes or no, or true or false. A Decision Diamond has two direction arrows, one pointing out the path the flowchart will follow if the answer to the question is yes (or true) and the other pointing out the path that flowchart will follow if the answer to the question is no (or false).

LABEL CONNECTORS



Label connectors are used when it is inconvenient, because of the layout or complexity of a flowchart, to show a lengthy direction arrow between two points that are logically connected. An arrow entering a label connector shows where a flowchart is to continue to. An arrow exiting from a connector shows where the program is to continue from. The letters or numbers inside the connector give a name, or label, to the connector.

Sometimes, a flowchart may be written on two or more pages. As a result, a label connector may send you to a place on the page you are reading or to another page. As a result, two different shapes are used for label connectors.

ON-PAGE LABEL CONNECTOR

OFF-PAGE LABEL CONNECTOR

Now that label connectors have been explained, we can use them to improve our flowchart. Note that in the flowchart in Figure B the three operation symbols contain almost identical operations. One says "Buy Cadillac", another says "Buy Dodge" and the last says "Buy Volkswagen". These can all be generalized to "Buy Car". This is all the information necessary for the decision making process, since you are already in the proper showroom when the decision to "Buy Car" is made. We can thus use label connectors to shorten the flowchart. Here is how the new flowchart might look:

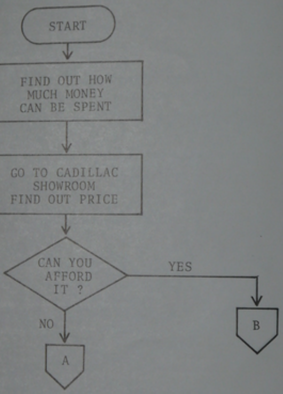


FIGURE C

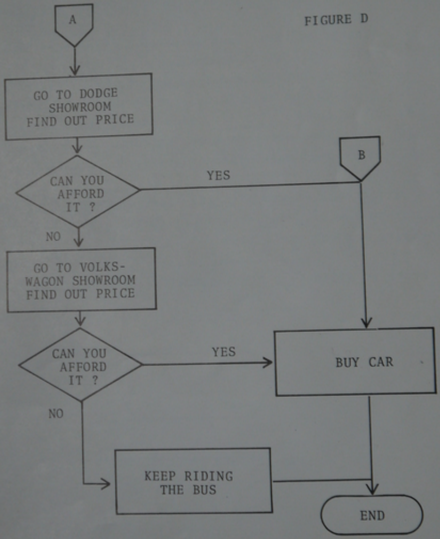
JUMPING

The flowchart in Figure D also illustrates the technique of "jumping". Jumps are used when you are at one point of a flowchart and wish to skip over several operations and go directly to another operation.

When writing programs, you will find that jumps are used in the decision making process.

The continuing flowchart might look as follows:

FIGURE D



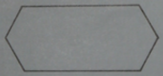
Because of the order in which the flowchart takes you to showrooms, it forces you to buy the most expensive car that you can afford. Realistically, buying a car because of the price tag is not a very good idea. The best car is not necessarily the most expensive one, it is the one that best suits your needs. Consideration should be given to: styling, mileage, safety, performance, seating capacity, trade-in-value, etc. After you've found out this information, you would then buy the most expensive car you could afford that best suits your needs. The flowchart should then be re-written to include these considerations.

First, let's list a set of rules to follow when writing the flowchart.

1. Arrange to visit each showroom in the order of expected price (highest to lowest).
2. Enter a showroom and find out the price of the car.
3. If it cannot be afforded, go to next showroom.
4. If it can be afforded, find out if it meets other criteria.
5. If it does not meet other criteria, go to next showroom.
6. If it does meet other criteria, buy the car.

By visiting each showroom in the order of expected price, from highest to lowest, you are assured that you will buy the most expensive car you can afford.

In order to write this new flowchart, we must introduce a new symbol:



PRE-DEFINED PROCESS SYMBOL

This symbol represents a group of operations (or program steps) that are specified elsewhere in a flowchart. If you encounter the symbol while reading a flowchart, read the group of operations that it represents, then come back to where you left off and continue reading the main body of the flowchart.

With this symbol and the plan just discussed, we can write a comprehensive flowchart.

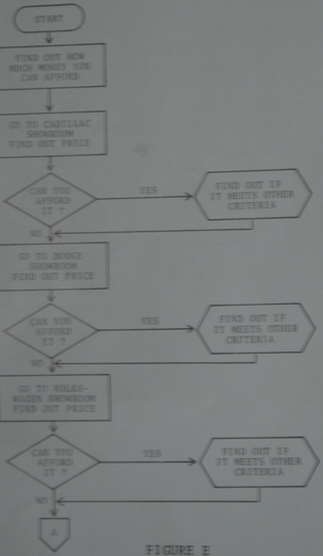


FIGURE E

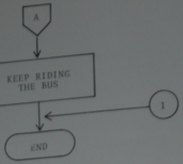


FIGURE F

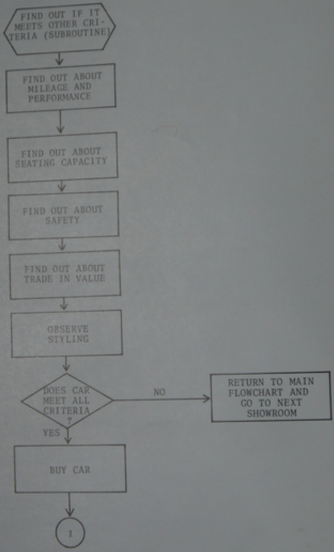


FIGURE G

SUBROUTINES

This flowchart has introduced an entirely new concept, subroutines. A subroutine is a small flowchart (or program) that is part of a larger flowchart (or program). The subroutine shown here was used to determine if a car met the criteria other than price. If it did, the subroutine directed you to purchase the car. If it didn't, the subroutine directed you to return to the point from which you left the main program. This 'return' feature is what distinguishes a subroutine from a jump. A jump takes the program from point A to point B. A subroutine takes the program from point A to point G and then at some pre-determined time takes the program back to point A. As illustrated by this flowchart, the program can go to a subroutine several times from several different parts of a program and always return to the point from which it left. When a computer reads an instruction to go to a subroutine, it records the point to which it should return. As a result, a program can branch to a subroutine several times and always return to the place from which it left.

A program is a series of instructions executed automatically by a computing device. The instructions are written by the programmer. To write a program, analyze the best way to solve the problem manually, then execute the problem once, depressing each key in the order in which the problem would be solved. If, to run the problem manually, you had to enter three numbers and execute thirty instructions, a program will reduce your work to entering three numbers on the same key.

WRITING A BASIC PROGRAM

For example, let's say that you would like to write a program to solve the equation $\frac{A \times B \times C}{D} = E$. This means that you want to be able to enter any four numbers, A,B,C and D, and have a program automatically multiply A times B times C then divide the product by D. If A=2, B=3, C=4 and D=5, therefore you would like a program to compute: $\frac{2 \times 3 \times 4}{5} = 4.8$

The first thing to do is determine how to solve the problem manually.

Problem	Solution
$\frac{2 \times 3 \times 4}{5} = 4.8$	2 x 3 x 4 ÷ 5 =

We could generalize this problem for any four variables A,B,C and D as follows:

Problem	Solution
$\frac{A \times B \times C}{D} = E$	A X B X C ÷ D =

To generalize the solution, letters (or variables) were substituted for the numbers 2, 3, 4 and 5.

Now, how would a program be written to compute $\frac{A \times B \times C}{D} = E$? The basic solution was given above, in the solution column.

A x
B x
C ÷
D =

There are additional steps to take in order to enter this program into the calculator. However, the logic of how it works is complete.

Let's go over these additional steps required for entering a program. Looking at the program, there are four places where a variable is to be entered. However, the program cannot be entered touching 2 [X] 3 [X] 4 [÷] 5 [=], as the variables will constantly change. There has to be a single instruction that will tell the machine to stop and allow entry of the next variable. The instruction that tells the machine to stop is logically called [STOP] and should be inserted in place of the variables in our program.

So far the program instructions are as follows:

INSTRUCTION	EXPLANATION
STOP	enter A
x	
STOP	enter B
x	
STOP	enter C
÷	
STOP	enter D
=	compute E

Notice that the last step says "compute E", not "compute and print E". A program will never execute anything unless it is specifically instructed what to do. In this case, it will not print anything unless you instruct it to print. Obviously, you must print the answer and you will undoubtedly want to print the entries as well. The complete program looks like this.

Program Coding Form

PROGRAM TITLE ____________ CALCULATOR MODEL NO. ____________

PROGRAMMER ____________

PAGE ______ OF ______ DATE ____________

000	STOP	Enter A	25			50		
01	PRINT	Print A	26			51		
02	X		27			52		
03	STOP	Enter B	28			53		
04	PRINT	Print B	29			54		
05	X		30			55		
06	STOP	Enter C	31			56		
07	PRINT	Print C	32			57		
08	÷		33			58		
09	STOP	Enter D	34					
10	PRINT	Print D	35					
11	=	Compute E	36					
12	PRINT	Print E	37					
13	SPACE		38					
14	JUMP		39					
15	0		40					
16	0							
17	0							
18								

The last four steps ([JUMP] 000) instructs the calculator that the program has been completed and that it should go back to the first step to prepare for new variables.

Step 013, [SPACE], instructs the program to skip a space each time it computes an answer. This makes the answers easier to see. This is called 'formatting" the output.

NOTE: The first step in a program is 000, not 001. This is common logic in any computing device. Therefore, the last available step is <u>one less</u> that the memory size of your calculator.

Now that the program is finished, let's enter it into the calculator. To start, find the [MANUAL] and [PROG] keys. The normal condition, [MANUAL], allows you to run both manual calculations and programs. To enter a program touch [MANUAL], touch [PROG], enter the program steps, then touch [MANUAL] again. The tape print out will list as follows:

```
000....
000....    ST
001....    P
002....        ×
003....    ST
004....    P
005....        ×
006....    ST
007....    P
008....        +
009....    ST
010....    P
011....        =
012....    P
013....    LS
014....    J
015....          0
016....          0
017....          0
```

Now we're ready to run the program.

First, touch the [RUN] key. Then enter the variables in order, touching the [RUN] key after each entry. When you're finished, the answer will print automatically. Try this:

ENTRY		TAPE
	RUN	2.e
2	RUN	3.e
3	RUN	4.e
4	RUN	5.e
5	RUN	4.80

Easy? Now try some new numbers. Enter 10, 12, 14 and 16. You should get an answer of 105. Enter 5, 2, 12 and 3. You should get an answer of 40. Enter any four numbers you like, and see what the answer is. Congratulations! You've just entered and run your first program.

RULES OF BASIC PROGRAMMING

Three rules to follow can be established when writing a program. These are the rules that were followed when writing the previous program.

1. ANALYZE THE PROBLEM and decide the best way to solve it manually.
2. WRITE DOWN THE SOLUTION.
3. INSERT [STOP], [PRINT] AND [SPACE] INSTRUCTIONS.

These rules should be easy to follow if you have carefully studied the way the calculator operates. First, choose a method of solving your problem that will work for any set of variables. Next, write down the solution in a way that uses a minimum number of key depressions. Finally, print the entries and answer, and insert the instruction [SPACE] as often as desired to set apart entries from answers and to separate different problems.

Let's write a short sample program to illustrate how these steps are followed. A very common business problem is one called Percentage Increase and Decrease. This problem computes the difference and percent of change between periods. For example, if 1250 units were sold in January and 1500 units in February then there has been an increase of 250 units or 20%.

1. ANALYZE THE PROBLEM

   Increase or Decrease:

   difference = current period - previous period

   Thus:

   $$\frac{\text{difference}}{\text{previous period}} = \text{\% of Increase or Decrease}$$

2. WRITE DOWN THE SOLUTION:

Entry	Key Depression	Explanation
	CLEAR ALL	
1500	+	Current Period
1250	÷	Previous Period
	-	
	T	Print 250, the amount of change
	EXCHANGE	
	%	Print 20%, the percentage of change.

3. INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.
   The completed form looks as follows.

Program Coding Form

CALCULATOR MODEL NO. ______ PROGRAM TITLE ______

PROGRAMMER ______ PAGE ______ OF ______

DATE ______

000	CLEAR ALL	
01	STOP	Enter current period
02	PRINT	
03	+	Add current period
04	STOP	Enter previous period
05	PRINT	
06	SPACE	
07	÷	Set up previous period as a dividend
08	-	Subtract previous period from current period
09	TOTAL	
10	PRINT	Print amount of change
11	EXCH-ANGE	Make previous period the divisor, and the difference the dividend
12	%	
13	SPACE	
14	PRINT	Print percentage change
15	SPACE	
16	SPACE	Separates each problem on the tape
17	SPACE	
18	SPACE	
19	JUMP	Return to the beginning to process a new set of data
20	0	
21	0	
22	0	
23		
24		

FORM 4019 PRINTED IN U.S.A.

SELECT MEMORY 1 2 3 -20- 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

JUMPS AND DECISIONS

In general, the steps of a program are executed in sequential order. When a [JUMP] instruction is executed, this sequential order changes. The program may go from the 52nd step to the 165th step, or from the 94th step to the 281st step, or from the 392nd step to the 8th step, and so on. The steps in between are ignored. The program simply starts executing from the new address.

Any program step within the memory size of the calculator can be addressed by a Jump instruction.

Jumps fall into two categories, conditional and unconditional. Unconditional jumps simply take a program from one address to an address designated by the jump instruction. For example, if the calculator encounters a set of instructions saying, "Jump to Step 23", the jump is executed in all cases, unconditionally. Conditional jumps, on the other hand, will jump from step 28 to 48 only if some condition is fulfilled. Therefore, conditional jumps are used when a program has to make a decision.

The calculator can make four different types of decisions. All concern the number in the keyboard register. If we call this number X, then the four decisions are:

CONDITION	INSTRUCTION
1. Is X positive?	Jump if Positive [JUMP] [+]
2. Is X negative?	Jump if Negative [JUMP] [-]
3. Is X zero?	Jump if Zero [JUMP] [=]
4. No X entry?	Jump if No Entry [JUMP] [AX]

In each case, if the statement is true, the program will jump. If the statement is false, the program will pass through and go to the next step.

To see how jumps are used, write a program to test if X is greater than zero, less than zero, or equal to zero.

1. If X is greater than zero, Print X and then print 12345.
2. If X is less than zero, print X and then print a negative 67890.
3. If X equals zero, print X and then 99999.

First analyze the problem and flowchart it.

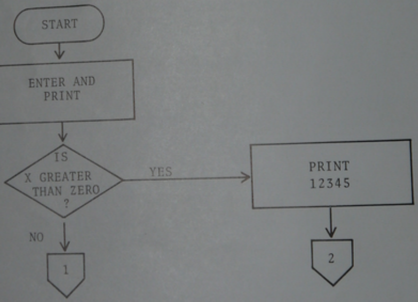


FIGURE H

Now that we have written the flowchart, we can write the program and insert [STOP], [PRINT] and [SPACE] instructions. Conditional jumps will be inserted to replace the decision diamonds.

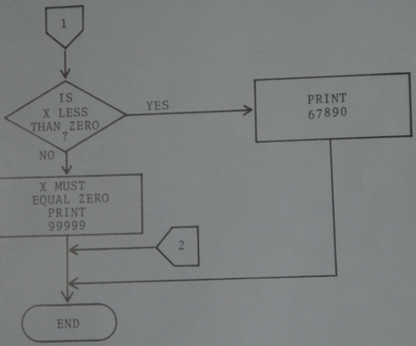


FIGURE I

Program Coding Form

CALCULATOR MODEL NO.

PROGRAMMER

DATE

PROGRAM TITLE

PAGE   OF

000	STOP	Enter x	25	1	Routine used when x>0	50	0		75		
01	PRINT		26	1		51	0		76		
02	JUMP		27	2		52			77		
03	+		28	3		53			78		
04	LABEL	Jump if x is >0	29	4		54			79		
05	0		30	5		55			80		
06	1		31	PRINT		56			81		
07	JUMP		32	SPACE		57			82		
08	-		33	J		58			83		
09	LABEL	Jump if x is <0	34	0		59			84		
10	0		35	0		60			85		
11	2		36	0		61			86		
12	9	If program reaches this	37	LABEL	Routine used when x<0	62			87		
13	9	step, then x=0. Since	38	0		63			88		
14	9	x=0, print 99999.	39	2		64			89		
15	9		40	6		65			90		
16	9		41	7		66			91		
17	PRINT		42	8		67			92		
18	SPACE		43	9		68			93		
19	JUMP		44	0		69			94		
20	0		45	CHANGE SIGN		70			95		
21	0		46	PRINT		71			96		
22	0		47	SPACE		72			97		
23	LABEL		48	JUMP		73			98		
24	0		49	0		74			99		

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

FORM 409 PRINTED IN U.S.A.

This program used two labels, label 01 and label 02. A label is simply a symbolic address. For more information, please refer to the explanation of [LABEL] in the Advanced Operation Instruction Manual.

This problem involved tests and jumps. However, most programs require that data be processed before it can be tested. For example, write a program in which a number X is entered. If X is less than or equal to 500, the program will multiply X by 2 and print the result. If X is greater than 500, the program will divide X by (-2) and print the result. In either event, after printing the result, the program will go back to the first step and prepare to accept another X value.

While this problem is very simple, use the three basic steps to solve it.

1. ANALYZE THE PROBLEM. decide the best way to solve it manually.

   To solve this problem, subtract X from 500 to determine its size. Then either multiply X by (2) or divide X by (-2), depending on the result of the operation. Since in subtracting 500 from the entry we will destroy it, we must temporarily store X in a memory while performing the test. After the test has been made, we can recall X from the memory and properly process it.

2. WRITE DOWN THE SOLUTION. This program involves both decisions and calculations; so it is best to construct a flowchart.

   The next step is to write the solutions to the two routines that will compute either 2 times X or X divided by -2.

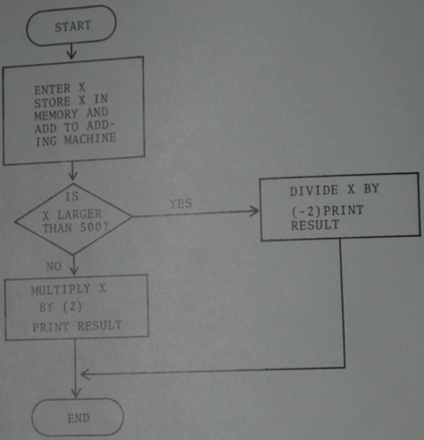

START
ENTER X
STORE X IN
MEMORY AND
ADD TO ADD-
ING MACHINE
IS
X LARGER
THAN 500?
YES
DIVIDE X BY
(-2)PRINT
RESULT
NO
MULTIPLY X
BY (2)
PRINT RESULT
END

FIGURE J

	Entry	Key Depression	Explanation
		T	Clear adding machine
		*	and memory
A. To determine the relative size of X	X	+	Add X to adding machine
		M+	and to memory
	500	-	Subtract 500 from X
		T	Bring (X-500) to keyboard register
		(Test Total)	
B. X is less than 500		*	
		X	Recall X from memory and multiply it by 2.
	2	=	
C. X is greater than 500		*	
		÷	
	2	CHANGE SIGN	Recall X from memory and divide it by (-2).
		=	

3. INSERT [STOP], [PRINT],[SPACE] AND [JUMP] INSTRUCTIONS.

We can now write the entire program by following the flowchart, stringing together the above short routines and inserting [STOP], [PRINT], [SPACE] and [JUMP] instructions.

Program Coding Form

CALCULATOR MODEL NO. ____________ PROGRAM TITLE ____________

PROGRAMMER ____________

DATE ____________ PAGE ______ OF ______

Step	Key	Comment
000	TOTAL	
01	*	
02	SPACE	
03	STOP	Enter X
04	PRINT	
05	+	
06	M+	
07	5	
08	0	
09	0	
10	-	
11	TOTAL	Bring X-500 to keyboard
12	JUMP	
13	+	If X>500, Jump. If
14	CLEAR ALL	X<500, go to next step
15	0	
16	1	
17	*	Recall X, clear memory
18	X	
19	2	X<500. Multiply it by 2.
20	=	
21	PRINT	
22	JUMP	
23	0	
24	0	
25	0	
26	LABEL	Start of routine to process X if X>500.
27	0	
28	1	
29	*	Recall X. Clear memory
30	÷	
31	2	Divide X by (-2)
32	CHANGE SIGN	
33	=	
34	PRINT	
35	JUMP	Return to beginning of program.
36	0	
37	0	
38	0	
39		
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

FORM 409 PRINTED IN U.S.A.

This program is now complete. To enter it, simply depress [MANUAL], [PROG], [CLEAR], enter the program steps, depress [MANUAL], [RUN], then enter the values. The answers will print.

USING CONSTANTS IN A PROGRAM

Note that when executing the previous program that steps 007, 008 and 009 generated the constant 500. You may generate any constant within the 14 digit capacity of the calculator. Each digit includ-ing sign and decimal point takes one step. (12 requires two steps, 1.2 requires three steps and (-1.2) requires four steps.)

THE THREE BASIC RULES OF PROGRAMMING

Now that several programs have been written, the basic rules of pro-gramming can be revised to allow entering of jumps. The expanded set of rules will be used throughout the rest of this book.

1. ANALYZE THE PROBLEM. Decide the best way to solve it manually. If the program is in several parts, analyze and solve each part separately.

2. WRITE DOWN YOUR SOLUTION. Unless the program is exceedingly simple, you shold write down both a flowchart of the problem and the method of solving each part of the problem.

3. INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS. Insert [STOP] [PRINT] and [SPACE] at the logical places in the program, as discussed earlier. Insert [JUMP] instruc-tions at the decision points of the flowchart.

CONSTRUCTING A BASIC PROGRAM

Now that you have seen how conditional jumps are used, let's put them to work in a business application.

As of this writing, social security is paid at the rate of 5.85% of annual earnings, up to a maximum earnings of $13,200. After a person's income for the year exceeds $13,200, social security is no longer deducted.

Let's write a program to compute social security on earnings. The program should operate as follows: The operator enters the gross earnings during the pay period (P); then enters the year-to-date earnings up to the most recent pay period (YTD). The program must compute and print the social security deduction (if any). If there is no deduction, the program should print zero.

1. ANALYZE THE PROBLEM

An excellent way to analyze a problem like this is to carefully select test values that cover every possible combination of year-to-date and weekly earnings. Consider the following chart:

Employee	Year-To-Date Earnings	Weekly Earnings
A.	$ 10,000	$ 250
B.	17,025	420
C.	13,000	350
D.	12,985	215
E.	13,200	385

'A' and 'B' are very straight-forward. 'A' has social security deducted from his wages and 'B' doesn't, as 'B' has already passed the $13,200 limit. 'C' has earned $13,000, so he must pay 5.85% of his next $200 in earnings. However, 'C' has earned $350 for the period so a portion of his earnings is exempt from social security and a portion is not. This means that our program will have to make a special allowance for 'C'. 'D' and 'E' are exactly the opposite of each other. 'D's' weekly earnings will bring the year-to-date earnings up to exactly $13,200. 'E's' year-to-date was exactly $13,200 before the weekly earnings were added.

When writing a program, carefully consider every conceivable combination of circumstances and prepare the program to properly process each potential problem.

Manually solving these problems is easy. For 'A' and 'D', we multiply the weekly earnings by 5.85%. For 'B' and 'E', we do nothing (print zero). For 'C', we first find the amount of which social security must be deducted (by subtracting the old year-to-date earnings from $13,200), then multiply this amount by 5.85%.

2. WRITE DOWN THE SOLUTION. The five problems can be solved as follows:

	Entry	Key Depression	Explanation
Employee A	250	X	
	5.85	%	
Employee B			No Deduction
Employee C	13200	+	
	13000	-	
		T	$13,200-YTD=amount on which to pay social security.
		X	
	5.85	%	
Employee D	215	X	
	5.85	%	
Employee E			No Deduction

As you can see, getting the program to choose the proper solution is necessary. Drawing a flowchart will give us direction for placing decisions. Remember that we are calling the year-to-date earnings 'Y-T-D' and the amount earned in the most recent pay period 'P'.

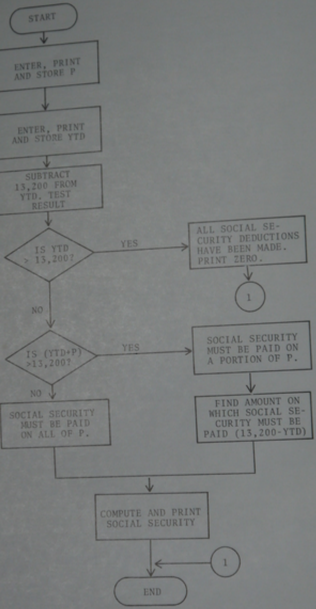


FIGURE K

Take a second to look at the question inside the first decision diamond. Note that another way of asking, "Is YTD greater than 13,200?", is to ask, "Is (YTD-13,200) greater than zero?". This is the way tests are made by the 900 Series. Programs must be written with this in mind.

You should also note the way the question in the second decision diamond is phrased. It asks, "Is (YTD+P) greater than 13,200?". This question was specifically phrased to allow for case D. 'D' was the person whose weekly earnings brought her year-to-date up to exactly 13,200. If the question in the decision diamond had been, "Is (YTD+P) less than 13,200?", or, "Is (YTD+P) greater than or equal to 13,200?", the program would not operate correctly. Its decision would have been to <u>not</u> deduct social security from 'D's' paycheck. This illustrates the need for drawing a flowchart and carefully selecting test values.

3. INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.

Program Coding Form

CALCULATOR MODEL NO. ______ PROGRAM TITLE ______

PROGRAMMER ______ PAGE ____ OF ____

DATE ______

Step	Key	Comments
000	CLEAR ALL	
01	STOP	Enter amount earned during pay period (P)
02	PRINT	
03	M+	Add P to memory
04	STOP	Enter YTD earnings
05	PRINT	
06	+	Accumulate YTD
07	1	
08	3	
09	2	
10	0	
11	0	
12	-	
13	S	YTD-13200
14	JUMP	
15	+	
16	LABEL	Jump if all FICA, soc.sec. paid.
17	0	
18	1	
19		Recall P
20	+	
21	T	(YTD+P)-13200
22	JUMP	Jump if soc. sec. is to be paid on P.
23	-	Continue if
24	LABEL	soc. sec. is to be paid on a
25	0	portion of P.
26	2	
27	M-	Memory now contains (13200 - YTD), the amt. on which soc. sec. must be paid
28	*	
29	X	
30	5	
31	.	
32	8	
33	5	
34	%	
35	PRINT	
36	SPACE	
37	JUMP	
38	0	
39	0	
40	0	
41	LABEL	
42	0	
43	1	
44	0	
45	PRINT	
46	SPACE	
47	JUMP	
48	0	
49	0	
50	0	
51	LABEL	
52	0	
53	2	
54	*	Recall P
55	X	
56	5	
57	.	
58	8	
59	5	
60	%	
61	PRINT	Soc. Sec.
62	SPACE	
63	JUMP	
64	0	
65	0	
66	0	
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

SHORTENING PROGRAMS

There are many programs that can be easily written within the number of steps available in your calculator. There is rarely any practical reason for shortening these programs. However, it is wise to practice step-saving techniques, so that you will be skilled at using them when confronted with an extensive program, which may initially utilize more steps than are available.

The first rule in eliminating steps is to make use of the keys that do the most work in the least number of steps. For example, when using one memory frequently, it is often best to use [SELECT M] and the six keys related to it ([M+], [M-], [=+], [=-], [◇] and [*]. These six keys all take one step to access the memory. By contrast, [→M/IN] and [←M/OUT] take three steps to store or recall data and four steps to accumulate data. Another economical set of keys are those relating to the add register ([+], [-], [S] and [T]). You might find that this is the most convenient 'memory' available. Each function takes just one step. Finally, be sure that you understand the functions of [AX], [N], [%] and [EXCHANGE]. These were carefully designed to give you a variety of automatic functions and proper use of these keys can save many steps and memories.

In this manual other methods of saving steps and registers are pointed out. Let us re-examine the social security program to see what can be done to shorten it. The program already takes advantage of the add register, the [SELECT M] functions and the percent key. It can be shortened even further, however, by some re-writing.

This program has three separate problems to consider:

1. Pay social security on the entire week's pay.
2. Pay social security on a portion of the week's pay.
3. Pay no social security at all.

They have been handled by writing three separate routines that handle each situation and then jump back to the beginning of program.

We can save a number of steps by combining parts of the routines, and letting some instructions operate for more than one routine.

Here is a simple example. It just saves two steps but explains the concept. At label 01 you will find the routine that prints zero when no social security is deducted. The steps are:

Step Number	Key Depression
045	PRINT
046	SPACE
047	JUMP
048	0
049	0
050	0

Steps 035 - 040 are:

035	PRINT
036	SPACE
037	JUMP
038	0
039	0
040	0

Since steps 045 - 050 are identical to steps 035 - 040, two steps can be saved by the following change:

Step Number	Key Depression
045	0
046	JUMP
047	0
048	3
049	5

This will have no effect on the answers that print. The program had been printing zero at step 045. Now it will print zero at step 035. In either case, after printing the answer it will jump to step 000.

By using this philosophy, we can shorten the rest of the program.

Program Coding Form

CALCULATOR MODEL NO. ______ PROGRAM TITLE ______

PROGRAMMER ______

DATE ______ PAGE ______ OF ______

000	CLEAR ALL		25	0	of P	50	0		75		
01	STOP	Enter amount earned during	26	2		51	0		76		
02	PRINT	last pay period (P)	27	M-		52			77		
03	M+		28	LABEL		53			78		
04	STOP	Enter YTD	29	0		54			79		
05	PRINT		30	2		55			80		
06	+		31	*	Pay soc. sec. on amount in	56			81		
07	1		32	X	memory	57			82		
08	3		33	5		58			83		
09	2		34	.		59			84		
10	0		35	8		60			85		
11	0		36	5		61			86		
12	-		37	%		62			87		
13	S	(YTD-13200)	38	PRINT	Social Security	63			88		
14	J		39	SPACE		64			89		
15	+		40	JUMP		65			90		
16	LABEL	Jump if all soc. sec.	41	0		66			91		
17	0	paid	42	0		67			92		
18	1		43	0		68			93		
19		(Recall P)	44	LABEL	No soc. sec. to be paid	69			94		
20	+		45	0		70			95		
21	T	YTD+P-13200	46	1		71			96		
22	JUMP	Jump if soc. sec. is to be	47	0	Zero to print	72			97		
23	-	paid on P. Continue ifsoc.	48	JUMP		73			98		
24	LABEL	sec. is to be paid on portion	49	0		74			99		

FORM 403 PRINTED IN U.S.A.

SELECT MEMORY 1 2 3 338- 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

This program uses the same basic logic as the previous one. If you have any questions, please refer to the explanation of the earlier program.

Our 67 step program has been cut down to 52 steps by making some simple changes. It's obviously valuable to save 15 steps, but the percentage savings, 22%, is even more significant. Theoretically, the same techniques could help you cut a program that first took 575 steps down to 448 steps.

During the rest of this manual we will discuss many other techniques that will help you save steps. However, it is literally impossible to cover all techniques so the best thing to do is to constantly think of ways to shorten programs while writing them.

Gradually, you will build a mental library of step-saving methods, and eventually you will find yourself easily solving problems that you would have once thought to be far beyond your calculator's capabilities.

LOOPS

A common problem in business statistics is finding the standard deviation of ungrouped data. The formula for the standard deviation (SD) is:

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

Where ΣX is the sum of all variables, ΣX^2 is the sum of the squares of the variables and n is the number of variables.

Lets go through the three steps and see how easy it is to write a program for this formula.

1. ANALYZE THE PROBLEM. A good program for standard deviation should require the operator to enter each X-value just once. Each time an entry is made, the program should automatically accumulate ΣX, then accumulate ΣX^2 and n. The program should then be ready to accept the next X value.

   After the last value is entered, there should be an easy method to instruct the program to jump to a routine that will process the data and print the answer.

   Different problems will have different numbers of variables. The program should allow entry of an many values as necessary, then go to the part of the program that processes the data after the last variable has been entered.

2. WRITE DOWN THE SOLUTION. Following is a flowchart of a program that will allow entry of any number of variables and then process the accumulations.

This flowchart is a diagram of a simple procedure called a "loop". A loop is used when a part of a program is to be repeated several times before the program goes forward. for example, in a 100-step program in which steps 25-50 are to be repeated several times, the program should be written so that each time it reaches step 050 it would automatically loop back to step 025 and start again. After some pre-determined signal, it should jump out of the loop and carry out steps 041-099. Depending on how the program is written, a loop can be repeated as many times as necessary.

3. INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.

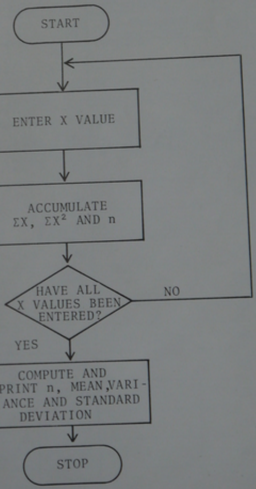


FIGURE L

Program Coding Form

CALCULATOR MODEL NO. ______________________ PROGRAM TITLE ______________________

PROGRAMMER ______________________

DATE ______________________ PAGE ______ OF ______

000	CLEAR ALL	Beginning of data entry loop.	25	AX		50			75		
01	STOP		26	=-		51			76		
02	JUMP		27	*		52			77		
03	AX		28	÷		53			78		
04	0		29	N	$\dfrac{\Sigma X^2 - \dfrac{(\Sigma X)^2}{n}}{n-1}$	54			79		
05	1		30	=		55			80		
06	4		31	PRINT	Variance	56			81		
07	PRINT	Entry	32	√‾	(Always floating)	57			82		
08	X		33	PRINT	Standard Deviation	58			83		
09	=+	Accumulate ΣX, ΣX^2 and n	34	SPACE		59			84		
10	JUMP		35	SPACE		60			85		
11	0		36	SPACE		61			86		
12	0		37	JUMP	Return to the beginning	62			87		
13	1		38	0		63			88		
14	SPACE	Beginning of data processing routine	39	0		64			89		
15	SPACE		40	0		65			90		
16	DEC SET	Insure 6 place accuracy	41			66			91		
17	6		42			67			92		
18	AX		43			68			93		
19	÷		44			69			94		
20	N	Number of entries	45			70			95		
21	PRINT		46			71			96		
22	=		47			72			97		
23	PRINT	Mean	48			73			98		
24	X		49			74			99		

FORM 409 PRINTED IN U.S.A.

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

This program will allow us to enter as many X-values as we like. Each time a variable is entered and [RUN] is touched, the program will accumulate ΣX, ΣX^2 and n, then loop back to step 000. After the last entry is made and [RUN] is touched, the operator merely touches [RUN] without making an entry. This will cause the program to break out of the loop and jump to step 014, where it will process the stored data.

USING THE INDIRECT MEMORY

A very common calculating problem is one called 'Percentage Distribution'. Several numbers are added together, then each individual number is divided by the sum of all the numbers to determine what percentage each is of the total amount.

First, let's see how this problem would be solved without using the indirect memory system. Assume that we have three numbers, 123, 456 and 789. We would like to write a program that will accumulate them, print a total and then print the percentage that each number contributes to the total. We have analyzed the problem in our discussion and we are ready to write down the solution.

	Entry	Key Depression	Explanation
		CLEAR ALL	To clear all registers
A. Routine to enter variables	123	+	To add the numbers
		→M/IN 01	To store for later use
	456	+	To add the numbers
		→M/IN 02	To store for later use
	789	+	To add the numbers
		→M/IN 03	To store for later use

B. Routine to process variables.	←M/OUT	01	Recall first entry
	÷		Establishes the total as a constant divisor
	T		
	%		First percentage
	←M/OUT	02	Recall second entry
	%		Second percentage
	←M/OUT	03	Recall third entry
	%		Third percentage
	AX		Total of percentages
	N		Number of entries

This is a very straight-forward approach, so we could now write the program by inserting [STOP], [PRINT] and [SPACE]. However, if we do, we won't be able to use the program unless there are exactly three entries.

We must design a loop, as we did in the previous problem, to handle any number of variables. Basically, this loop should store the first entry in memory 1, the second entry in memory 2 and so on. Such a program would be able to accept any number of entries, up to the capacity of the calculator. Each entry should be added to the adding machine, then stored in a separate memory. After the last entry is made, the program should automatically leave the loop that stored all the variables, then enter another loop that recalls the variables and divides each by the total to find the percentages. After recalling and processing the last variable, the program must automatically leave the second loop, go back to step 000 and prepare to accept a new set of data.

Flowchart this method.

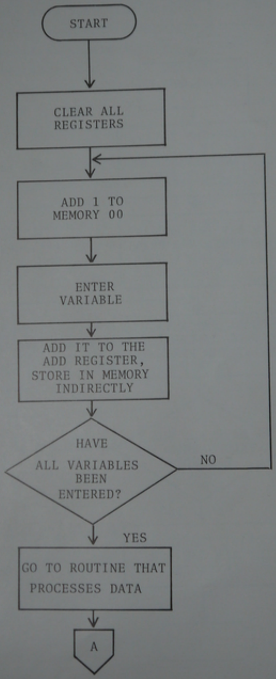

START
CLEAR ALL REGISTERS
ADD 1 TO MEMORY 00
ENTER VARIABLE
ADD IT TO THE ADD REGISTER, STORE IN MEMORY INDIRECTLY
HAVE ALL VARIABLES BEEN ENTERED?
NO
YES
GO TO ROUTINE THAT PROCESSES DATA
A

FIGURE M

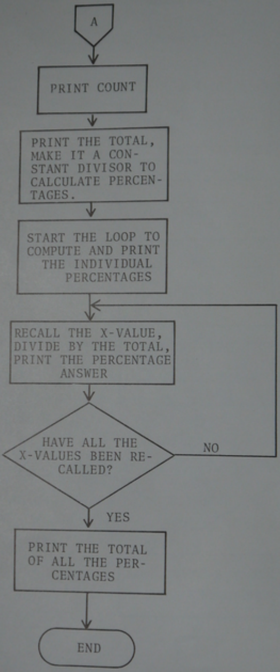

A
PRINT COUNT
PRINT THE TOTAL, MAKE IT A CONSTANT DIVISOR TO CALCULATE PERCENTAGES.
START THE LOOP TO COMPUTE AND PRINT THE INDIVIDUAL PERCENTAGES
RECALL THE X-VALUE, DIVIDE BY THE TOTAL, PRINT THE PERCENTAGE ANSWER
HAVE ALL THE X-VALUES BEEN RECALLED?
NO
YES
PRINT THE TOTAL OF ALL THE PERCENTAGES
END

FIGURE N

On two separate occasions, the program must automatically leave a loop. It will be simple to design the program so that it leaves the first loop after the last entry is made. A 'Jump If No Entry' will work perfectly, as it did in the program for standard deviation. We will have a problem with the second loop. The entire loop is automatic and so a 'Jump If No Entry' will not work. To do this we will have to construct an item counter.

The 900 Series has two separate, automatic counters. If you have studied the calculator, you will understand why an item counter (or N-count) is used. In the first loop, our program will automatically count the number of entries as we add each entry to the adding machine. All we must do in the second loop is "reverse" that counter, count backwards from the number of entries to zero and then leave the loop when the "reverse count" reaches zero.

The following program uses the indirect register, the reverse counter, and the other techniques that have been discussed.

Program Coding Form

CALCULATOR MODEL NO. ____________ PROGRAM TITLE ____________

PROGRAMMER ____________

DATE ____________ PAGE ______ OF ______

Step	Key	Remarks
00	CLEAR ALL	Clear all memories
01	SPACE	
02	LABEL	Beginning of data entry
03	0	loop
04	2	
05	SELECT M	Increment pointer
06	+	
07	STOP	Enter variable
08	JUMP	
09	AX	
10	LABEL	If no entry, jump to
11	0	process data.
12	1	
13	PRINT	Entry
14	+	Accumulate variable
15	→M/IN	
16	INDIR	Store variable indirectly
17	JUMP	
18	LABEL	
19	0	Return to beginning of
20	2	this loop.
21	LABEL	
22	0	Beginning of data process-
23	1	ing
24	SPACE	
25	N	Print number of entries
26	PRINT	
27	S	
28	PRINT	Print total of entries.
29	÷	Establish total as a
30	T	constant divi- sor.
31	=	
32	SPACE	
33	N	Initialize counter to
34	+	stop this loop after last
35	0	variable is recalled from
36	→M/IN	memory. Initialize
37	0	pointer reg- ister to re-
38	0	call first entry.
39	LABEL	
40	0	Data processing
41	3	
42	SELECT M	Increment pointer
43	+	
44	←M/OUT	Recall entry
45	INDIR	Compute entry's per-
46	%	centage of total
47	PRINT	
48	1	
49	-	
50	S	
51	JUMP	Jump if more processing
52	+	required.
53	LABEL	
54	0	
55	3	
56	SPACE	Processing completed
57	SPACE	
58	SPACE	
59	AX	
60	Print	Accumulated percentages
61	JUMP	(proof)
62	0	
63	0	
64	0	
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

This program automatically created a counter by the addition of each variable, then it used the count to form a reverse counter that went back to zero. Also, the program created a constant divisor with the steps [÷] [T] [=]. The answer to this division is of no importance but the total was converted into a constant divisor in only three steps.

As you can see, the indirect memory is a very powerful system. This program took just 65 steps to store and recall as many memories as was necessary. To do this without the use of the "Pointer Register" would have taken considerably more steps.

PRINTING AN IDENTIFIER ON THE TAPE

The Percentage Distribution program will produce a tape that is somewhat difficult to read. The way to solve this problem is to create a counter in the program, and use it to identify the number of each entry and answer. The output may be either of the two following formats.

1.

```
      1.    Identifier
  123.00    (first entry)

      2.    Identifier
  456.00    (second entry)

      3.    Identifier
  789.00    (third entry)

 1368.00    (total of entries)
    3.00    (number of entries)

      1.    Identifier
    8.99    (1st percentage)

      2.    Identifier
   33.33    (2nd percentage)

      3.    Identifier
   57.68    (3rd percentage)

  100.00    (total of percentages)
```

2.

```
      3.    (number of entries)
 1368.00    (total of entries)

      1.    Identifier
  123.00    (first entry)
    8.99    (first percentage)

      2.    Identifier
  456.00    (second entry)
   33.33    (second percentage)

      3.    Identifier
  789.00    (third entry)
   57.68    (third percentage)

  100.00    (total of percentages)
```

Either method is acceptable. The first method is simple to program and one that you can write.

The second method is not so obvious. Since each percentage is printed immediately after its corresponding entry, all variables must be entered before the first one can print. The entries will be printed during the second loop, as they are recalled from the memory.

We will have to change the previous program to get identifiers to print on the tape. A count is already being created in memory 00. This count can be recalled and printed during the second loop. To make the program print properly, steps 042-047 should be replaced with steps 042-053 below.

Step No.	Key Depression	Explanation
042	SELECT M	
043	+	Add 1 to memory 00
044	←M/OUT	Recall and print count from memory 00. This will create an identifier on the tape.
045	0	
046	0	
047	SPACE	
048	PRINT	
049	←M/OUT	Recall and print each entry.
050	INDIR	
051	PRINT	
052	%	Compute and print each entry's percentage of the total.
053	PRINT	

Next, step 013, which is a [PRINT] instruction, should be eliminated as it has been replaced by the [PRINT] instruction at step 051.

Because of the fine editing system available on the 900 Series, you will not have to re-enter this program. If the old program is in the program memory, change it to incorporate the new steps. The old routine was:

Step No.	Key Depression
042	SELECT M
043	+
044	←M/OUT
045	INDIR
046	%
047	PRINT

We can leave steps 042, 043 and 044 alone, erase step 045, then insert the new steps from 045 to 051. The program will automatically expand to allow room for the new steps. The [%], [PRINT], instructions that had been steps 046 and 047 will now be steps 052 and 053. Everything else will be adjusted accordingly.

1. Touch [JUMP] 045 [EDIT].
2. Touch [CLEAR] once to erase the [INDIR] instruction.
3. Enter the new steps, 00 [SPACE] [PRINT] [←M/OUT] [INDIR] [PRINT].
4. Touch [MANUAL] [JUMP] 013 [EDIT] [CLEAR] [MANUAL] to eliminate the [PRINT] instruction at step 013.

You might have wanted to correct step 013 before going on to step 045. However, that would complicate matters. When you eliminate step 013, the old step 014 becomes step 013, and so on. Step 045 would become step 044. Therefore it is always best to start with the last change and work backwards.

If you have followed the instructions properly, the tape of the editing instructions should look like this:

```
045....
         C
045....      0
046....      0
047....  LS
048....  P
049....  -M
050....  IN
051....  P
013....
         C
```

You may list out the new program by touching [LIST]. The tape should match this one:

```
000....  CA
001....  LS
002....  LA
003....      0
004....      2
005....  SE
006....     ÷
007....  ST
008....  J
009....   A×
010....  LA
011....      0
012....      1
013....     +
014....  -M
015....  IN
016....  J
017....  LA
018....      0
019....      2
020....  LA
021....      0
022....      1
023....  LS
024....   N
025....  P
026....   S
027....  P
028....     ÷
029....   T
030....     =
031....  LS
032....   N
033....     +
034....      0
035....  -M
036....      0
037....      0
038....  LA
039....      0
040....      3
041....  SE
042....     +
043....  -M
044....      0
045....      0
046....  LS
047....  P
048....  -M
049....  IN
050....  P
051....     %
052....  P
053....      1
054....     -
055....    S
056....  J
057....     +
058....  LA
059....      0
060....      3
061....  LS
062....  LS
063....  LS
064....   A×
065....  P
066....  J
067....      0
068....      0
069....      0
```

To run the program, enter the variables 123, 456 and 789. The printout should look like this:

```
      3.
   1368.

      1.
 123.00
   8.99

      2.
 456.00
  33.33

      3.
 789.00
  57.68


 100.00
```

Identifiers are extremely important, particularly when writing programs that print a great deal of information.

DEFINITION OF SUBROUTINES

Subroutines are powerful tools that have many uses. If a program has a routine that is to be used in several different places, writing a subroutine can save many program steps. Insert the steps [LABEL] (digit) (digit) in front of the subroutine and insert [RETURN] after the last step. (Remember that (digit) (digit) means and two-digit integer such as 01 or 99). Place the subroutine at the end of the main body of the program or in any area that makes sense. The instructions [GO SUB] (digit) (digit) will send the main program to the subroutine, and the [RETURN] instruction at the end will send the program back to the first step after [GO SUB] (digit) (digit).

PROGRAM USING A METRIC CONVERSION SUBROUTINE

The following is an example of a program that uses a subroutine to save 43 steps. Assume that we must write a program that converts yards to meters and cubic yards to cubic meters. Specifically, there are a number of boxes with measurements given in meters. The program should convert each side to its equivalent in yards, then find the volume of the box in both cubic meters and cubic yards.

To find the volume of a box with length L, width W and height H, the formula is simply; Volume = L x W x H.

To convert a number from meters to yards, it should be multiplied by 1.0936132983377.

1. ANALYZE THE PROBLEM
   Perhaps the most straightforward method is to enter each variable, store it in meters, convert it to yards, then store the converted number in a separate memory.

After the third variable is entered, converted and stored, the program can recall the three variables expressed in meters, find their product, print the answer, recall the variables in yards, find the product and print it. Finally, the program should clear all the registers and prepare to accept a new set of data.

2. WRITE DOWN THE SOLUTION

Flowchart the basic logic flow of the program, then incorporate the subroutine technique to shorten the number of steps that the program will use.

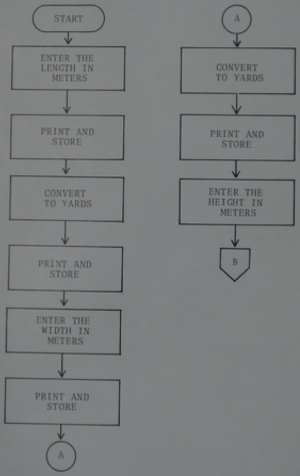


FIGURE 0

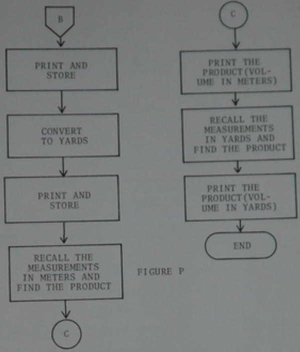
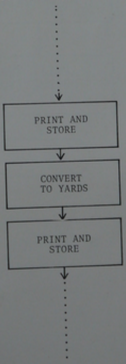


FIGURE P

By studying this flowchart the need for a subroutine becomes obvious. One series of instructions is repeated three times.



The flowchart will be rewritten using the subroutine technique.

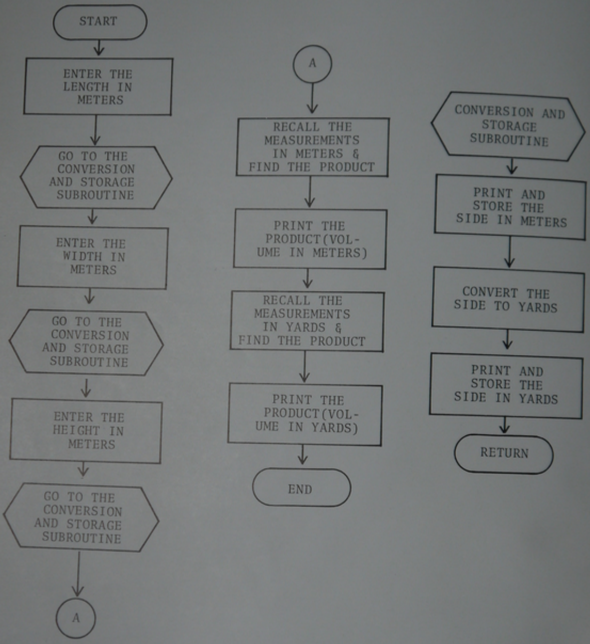

START
ENTER THE LENGTH IN METERS
GO TO THE CONVERSION AND STORAGE SUBROUTINE
ENTER THE WIDTH IN METERS
GO TO THE CONVERSION AND STORAGE SUBROUTINE
ENTER THE HEIGHT IN METERS
GO TO THE CONVERSION AND STORAGE SUBROUTINE
A
A
RECALL THE MEASUREMENTS IN METERS & FIND THE PRODUCT
PRINT THE PRODUCT(VOL-UME IN METERS)
RECALL THE MEASUREMENTS IN YARDS & FIND THE PRODUCT
PRINT THE PRODUCT(VOL-UME IN YARDS)
END
CONVERSION AND STORAGE SUBROUTINE
PRINT AND STORE THE SIDE IN METERS
CONVERT THE SIDE TO YARDS
PRINT AND STORE THE SIDE IN YARDS
RETURN

FIGURE Q

The flowchart indicates that there are only two major routines to be written.

a. Routine to Convert Meters to Yards and Store Both Numbers

Entry	Key Depressions	Explanation
5 (meters)	→M/IN 01	(store meters)
	X	
1.0936132983377	=	(convert to yards)
	→M/IN 02	

Now write the routine that calculates the volumes. The subroutine is to be used three times.

When a subroutine is used several times to store data that will be used later on in the main program, the Pointer Register should be used to load the memories. (The same logic applies to loops). The subroutine looks like this:

CONVERSION AND STORAGE SUBROUTINE

Key Depression	Explanation
PRINT	Print side in meters
SELECT M	
+	Add 1 to memory 00
→M/IN	
INDIR	Store this side in meters
X	
1	
.	
0	
9	
3	
6	
1	

Key Depression	Explanation
3	
2	
9	
8	
3	
3	
7	
7	
=	Convert this side to yards
SELECT M	
+	Add 1 to memory 00
→M/IN	Store this side in yards
INDIR	

If this routine is used three times, it will load the six measurements into six consecutive registers. If memory 00 is cleared out at the beginning of the program, then L (in meters) will go to memory 1, L (in yards) will go to memory 2, W (in meters) will go to memory 3 and H (in yards) will go to memory 6. Assuming this is the case, write a routine that will find the volume of the box in both cubic meters and cubic yards.

b. | Key Depression | Explanation |
|---|---|
| ←M/OUT | |
| 0 | |
| 1 | L in meters |
| X | |
| ←M/OUT | |
| 0 | |
| 3 | W in meters |
| X | |
| ←M/OUT | |
| 0 | |
| 5 | H in meters |
| = | Volume in meters |
| ←M/OUT | |
| 0 | |
| 2 | L in yards |
| X | |
| ←M/OUT | |
| 0 | |
| 4 | W in yards |
| X | |
| ←M/OUT | |
| 0 | |
| 6 | H in yards |
| = | Volume in yards |

Incorporate both routines into one program that allows entry of three variables.

3. INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.

Program Coding Form

CALCULATOR MODEL NO. ____________ PROGRAM TITLE ____________

PROGRAMMER ____________

DATE ____________ PAGE ______ OF ______

Step	Key	Comments	Step	Key	Comments	Step	Key	Comments	Step	Key	Comments
000	CLEAR ALL	Clear memories	25	0		50	LABEL		75	SELECT M	Increment indirect
01	STOP	Enter length	26	5		51	0		76	+	pointer
02	GO SUB		27	=		52	1	Subroutine to convert and	77	→M/IN	
03	0		28	PRINT	Volume in meters	53	PRINT	store data.	78	INDIR	Store variable
04	1		29	←M/OUT	Calculate volume in yards	54	Select M	Increment indirect	79	PRINT	
05	STOP	Enter width	30	0		55	+	pointer	80	SPACE	
06	GO SUB		31	2		56	→M/IN	Store variable	81	RE-TURN	Return to point of
07	0		32	X		57	INDIR		82		departure in main program
08	1		33	←M/OUT		58	X		83		
09	STOP	Enter height	34	0		59	1		84		
10	GO SUB	Subroutine transfer	35	4		60	.		85		
11	0		36	X		61	0		86		
12	1		37	←M/OUT		62	9		87		
13	SPACE		38	0		63	3		88		
14	SPACE		39	6		64	6		89		
15	SPACE		40	=		65	1		90		
16	←M/OUT	Calculate volume in	41	PRINT	Volume in yards	66	3		91		
17	0	meters	42	SPACE		67	2		92		
18	1		43	SPACE		68	9		93		
19	X		44	SPACE		69	8		94		
20	←M/OUT		45	SPACE		70	3		95		
21	0		46	JUMP		71	3		96		
22	3		47	0		72	7		97		
23	X		48	0		73	7		98		
24	←M/OUT		49	0		74	=	Meters converted to yds.	99		

FORM 409 PRINTED IN U.S.A.

SELECT MEMORY 1 2 3 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

Notice the economy of steps achieved by using a subroutine ([GO SUB] and [RETURN] instructions).

The operating instructions for the program are:

1. Depress [PROG], [CLEAR], then enter the program.
2. Depress [MANUAL], [RUN].
3. Enter the length in meters and depress [RUN]. The length in yards will print.
4. Enter the width in meters and depress [RUN]. The width in meters and yards will print.
5. Enter the height in meters and depress [RUN]. The height in meters and yards will print.
6. The volume in meters will print, followed by the volume in meters and yards.
7. The program is ready to accept a new set of variables.

If you run this program for the variables (5,6,7) and (9.50, 4.41 and 4.71) you will get the following printout with decimal at FL:

```
               5. e                           9.50 e
5.4680664916885               10.389326334208
               6. e                           4.41 e
6.5616797900262                4.8228346456692
               7. e                           4.71 e
7.6552930883639                5.1509186351705


              210.                         197.32545
  274.669630056                258.09194453397
```

CONDITIONAL SUBROUTINES

The metric conversion program used an unconditional subroutine. The instructions [GO SUB] 01 sent the program to label 01 under all circumstances. The 900 Series also has the ability to perform conditional subroutines under the same conditions that it will perform conditonal jumps. The calculator tests the number in the keyboard register. It can go to a subroutine under any of the following conditions.

	INSTRUCTION
1. Is X Positive?	Branch if Positive [GO SUB] [+]
2. Is X Negative?	Branch if Negative [GO SUB] [-]
3. Is X Zero?	Branch if Zero [GO SUB] [=]
4. No X Entry?	Branch if no Entry [GO SUB] [AX]

In each case, if the statement is true, the program will go to the subroutine. If the statement is false, the program will pass through and go to the next step.

NESTED SUBROUTINES

If a program is calculating in a subroutine, it can jump to a second subroutine, go through its steps, return to the point it left off at in the first subroutine, then go back to the point it jumped from in the main program. This process is called "nesting subroutines". The 900 Series can nest subroutines five levels deep, which means it can jump from subroutine A to subroutine B to subroutine C to subroutine D to subroutine E, complete E, go back to D and complete D, then go back to C, B, A and finally return to the main program. The following diagram will explain this concept.

START

.

.

.

.

.

.

Main Program   Go Sub

0

1

.

.

End

LA

Subroutine #1   0

1

.

.

.

.

.

Go Sub

0
2
.
.
.
.
Return
.
.
.
.
LA

Subroutine #2 0
2
.
.
.
.
.
GO Sub
0
3
.
.
.
.
Return
LA

Subroutine #3 0
3
.
.
.

Go Sub
0
4
.
.
.
Return
LA

Subroutine #4 0
4
.
.
Go Sub
0
5
.
.
.
.
Return
LA

Subroutine #5 0
5
.
.
.
.
Return

ERROR CORRECTION SUBROUTINES

The program to compute standard deviation (refer to loops, page 40) will operate with one exception: what will happen if the operator enters an incorrect value and touches [RUN]? He will have to touch [CLEAR ALL] and start over again. There should be a method of correcting such errors.

This can easily be solved by writing an error-correction routine. Error correction routines are written by following the same three steps that we have been using to write complete programs.

1. ANALYZE THE PROBLEM
   If an incorrect X-Value was entered and [RUN] was touched, ΣX, ΣX^2 and n accumulations would be incorrect. If you have entered a wrong number, X, then the error correction routine must subtract X from ΣX, X^2 from ΣX^2 and 1 from the n count.

2. WRITE DOWN THE SOLUTION
   The following routine corrects the error and makes it obvious on the tape.

Key Depression	Explanation
	Enter wrong entry X
SPACE	
SPACE	
EXCHANGE	
CHANGE SIGN	
PRINT	Print both X and -X to signify error correction.
CHANGE SIGN	Restore the sign of X
X	Correct error
=-	
SPACE	
SPACE	

Because of the spaces and the red printing of X, the tape will clearly show that this is not an entry but an error correction.

The error correction routine can now be incorporated into the main program. To allow the operator to access it manually, label 99 will be inserted at the beginning of the routine. Touching [GO SUB] 99 or [JUMP] [LABEL] 99 [RUN] will take the program to the routine.

The complete program for standard deviation looks like the following one.

3. INSERT [STOP], [PRINT], [SPACE] AND [JUMP] INSTRUCTIONS.

Program Coding Form

CALCULATOR MODEL NO. ____________ PROGRAM TITLE ____________

PROGRAMMER ____________

DATE ____________ PAGE ______ OF ______

000	CLEAR ALL	Clear memories	25	AX		50	X		75		
01	STOP		26	=-		51	=-	Nullifies incorrect entry	76		
02	JUMP		27	*		52	SPACE		77		
03	AX		28	÷	$\frac{\Sigma X^2 - \frac{(\Sigma X)^2}{n}}{n - 1}$	53	SPACE		78		
04	0		29	N		54	JUMP		79		
05	1		30	=		55	0	Jump to continue data entries	80		
06	4		31	PRINT	Variance	56	0		81		
07	PRINT		32	√		57	1		82		
08	X	Accumulate X, X^2 and n	33	PRINT	Standard deviation	58			83		
09	=+		34	SPACE		59			84		
10	JUMP		35	SPACE		60			85		
11	0		36	SPACE		61			86		
12	0		37	JUMP	End of main program	62			87		
13	1		38	0		63			88		
14	SPACE		39	0		64			89		
15	SPACE		40	0		65			90		
16	DEC SET	Insures six place accuracy regardless of decimal setting	41	LABEL	Error correction routine	66			91		
17	6		42	9		67			92		
18	AX		43	9		68			93		
19	÷		44	SPACE		69			94		
20	N		45	SPACE		70			95		
21	PRINT	Count	46	EX	Recall entry	71			96		
22	=		47	CHANGE SIGN		72			97		
23	PRINT	Mean	48	PRINT	Red to identify	73			98		
24	X		49	CHANGE SIGN	Restore proper sign	74			99		

SELECT MEMORY 1 2 3 -70- 4 5 6 7 8 9

Use a paper clip to indicate the memory being addressed. Move as necessary.

FORM 404 PRINTED IN U.S.A.

The loading and operating instructions for this program are as follows:

1. Depress [MANUAL] [PROG] [CLEAR]. This brings the step pointer to step 000 and eliminates any old program steps that had been held in the program memory.
2. Enter the program, then touch [MANUAL] [RUN].
3. Enter each X-value.
4. If a mistake is made and [RUN] is touched, do the following:
   a. Touch [GO SUB] 99. The tape will space twice.
   b. The incorrect number will print in red. This will signify that a correction has taken place.
   c. The mistake is now corrected. Continue entering data.
5. After all variables are entered, touch [RUN] to see the following results:

   Count
   Mean
   Variance
   Standard Deviation
6. To find the standard deviation of a new set of data, enter new data.

Rockwell