1151
PROGRAMMABLE CALCULATOR

PROGRAMMING INSTRUCTIONS

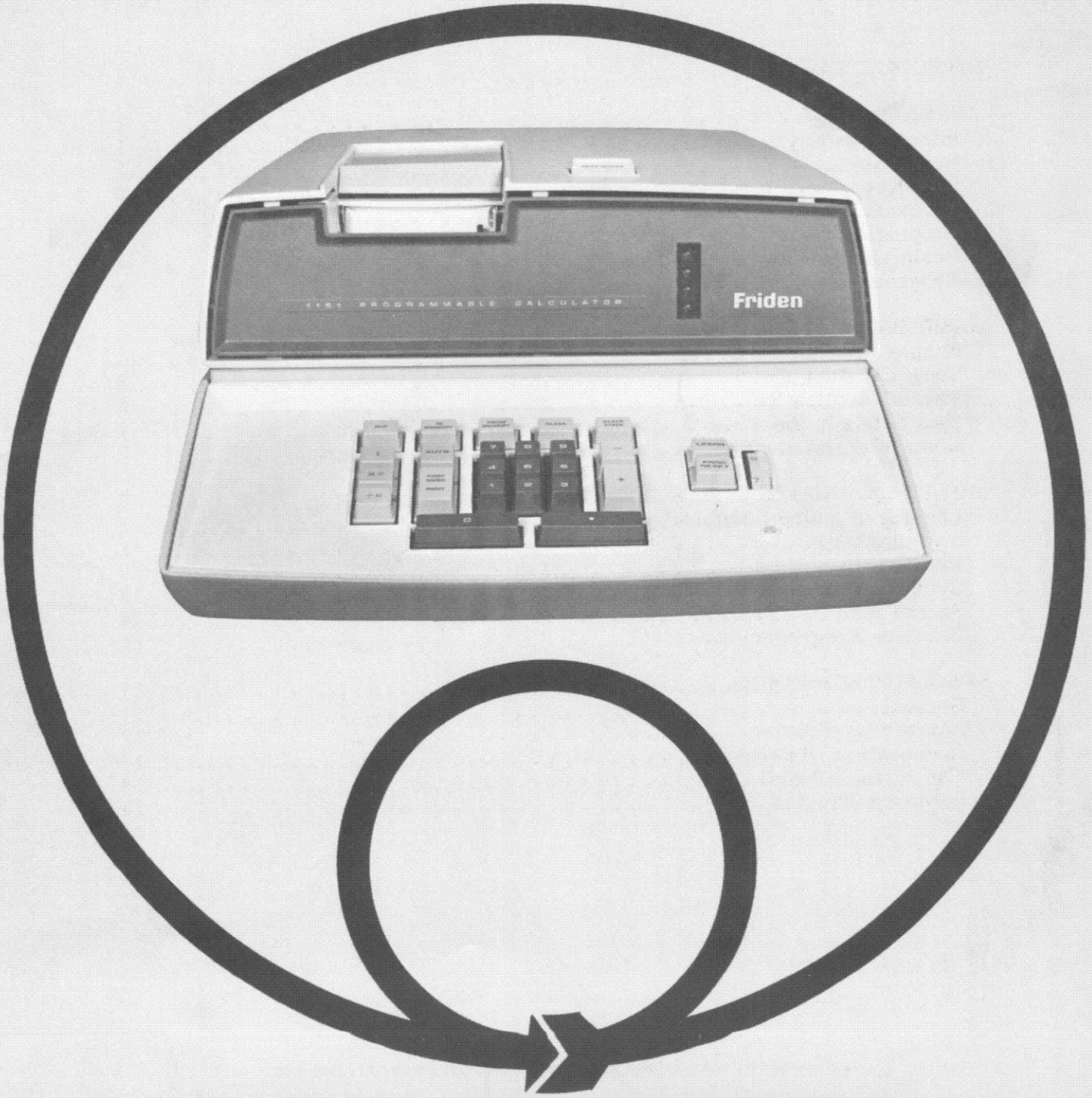


TABLE OF CONTENTS

Published by Friden Sales Promotion Department, Rochester, New York

INTRODUCTION

Welcome to the world of "customized calculators". The Friden* 1151 Programmable Calculator can be customized to your individual calculating requirements. The 1151 benefit that makes this possible is Program Learning.

Manual operation of the 1151 is covered in detail in SP 9474, 1151 Operating Instructions. That manual also contains basic information about the Program Learning capabilities of the 1151. In addition to the 1151 Operating Instructions, three application manuals are available: SP 9495, Statistics, SP 9496, Finance, and SP 9497, Science and Engineering. These booklets describe solutions to many problems for which the 1151 is particularly suited.

In this manual, 1151 Programming Instructions, a working knowledge of the manual operation of the 1151 is assumed. Let's start by discussing the 1151 keys that are particularly important to Program Learning.

GENERAL

THE INTERCHANGE KEY

The INTERCHANGE key interchanges the contents of the bottom two registers in the stack. Some calculations require that the divisor in a division problem be calculated before the dividend. This often results in the divisor being contained in R2 with the dividend in R1. The INTERCHANGE key allows us to exchange the contents of R1 and R2, thus placing the divisor in R1 so that we may properly divide. Added benefits of this powerful key are that a constant can be retained in the stack, or that two totals can be carried in the stack. Examples given in this manual are Payroll and Cost/Sell Quotations.

THE LEARN KEY

Depression of the LEARN key places the 1151 into a learning condition. Until the LEARN key is restored to its original position by depressing the PROGRAM RESET key, the 1151 will learn in sequence the functions required to perform the desired calculation.

The following commands are programmable:

Plus
Minus
Times Equals
Divide Equals
Interchange
Duplicate
To Memory
From Memory
Print
Stop

The manual functions, Clear, Clear Stack, and Stack Read-Out cannot be programmed.

These commands are self-explanatory, except, perhaps, for two: Print and Stop. You must instruct the 1151 as to which answers are to be printed on the tape during automatic operation by depressing FIRST NMBR/PRINT when they occur in R1. When programming an application, ask yourself at each step if you want the contents of R1 to print on the tape during automatic operation.

A Stop Command in a program sequence automatically stops automatic operation. It is placed in a program whenever a manual keyboard entry is made while the 1151 is learning an application. Stop Commands are used for three reasons:

1. To allow for manual entries of variable numbers into the program.
2. To allow for manual calculations, the answer, or answers, to which to be used in the program.

*A Trademark of THE SINGER COMPANY

3. To separate a portion of a program from the remainder of the program instructions.

The first reason for using a Stop Command is most important. What use would there be for programming if you could not use the learned program with a different set of variables?

The second reason for using a Stop Command is important particularly when an application requires more than 30 program steps. Normally, the Stop Command would be the first step in the program so that the first part of the application could be done manually and the program entered to complete the problem. Though it takes a little more programming finesse, we could design our program so that we have a Stop Command somewhere in the middle of a sequence of instructions for such manual calculations.

The third function of a Stop Command points up the greatest advantage that the 1151 has over any other programmable calculator to date: the capability of storing several programs at one time, to be used individually or in combination, at the option of the operator! An example of such a program is SP 9541, Coefficient of Linear Correlation. Three programs in one are used. The operator uses the first and second programs to obtain the sum of x, the sum of x^2, and the xy values. Then the operator uses only the first program to compute the sum of y and the sum of y^2. Finally, the operator uses only the third program to calculate the square root of the value obtained by a manual calculation done at the Stop Command between the second and third programs! Let's face it. Multiple-part programs can be very complicated to devise. But they do not have to be complicated to use!

As stated on page 1, there are three manual 1151 functions that cannot be programmed.

Clear
Clear Stack
Stack Read-Out

The functions Clear and Clear Stack can be accomplished through special programming. For example:

To Clear R1:

1. DUP
2. - (0 in R1)

To Clear Stack:

1. DUP
2. -
3. x=
4. x= (0 in R1, R2, R3, R4)

The function Stack Read-Out can be accomplished only by destroying the contents of 2 of the 5 registers in the calculator.

THE AUTO KEY

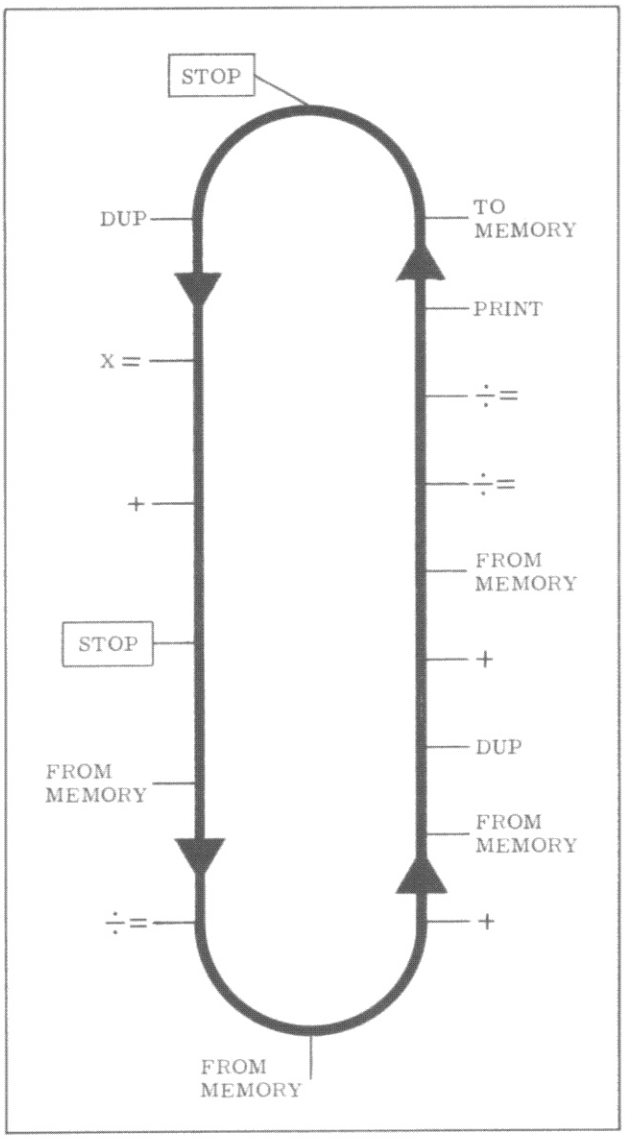
At any point where a manual entry must be made in an established program, the number is indexed on the keyboard and entered into the calculator by depressing the AUTO key. The 1151 will then continue with the automatic program using the new number. When the AUTO key is depressed, the number indexed on the keyboard is printed on the tape and identified by the symbol "F".

THE PROGRAM RESET KEY

Depression of the PROGRAM RESET key cycles the 1151 program memory back to the first step in the calculating procedure. If the first step in the learned program was a manual keyboard entry (Stop Command), the 1151 will stop, ready to accept a new manual entry. If the first step in the program is not a manual entry, the 1151 will automatically carry out all the steps up to the first point where a manual entry is to be made and then stop.

LOOPING

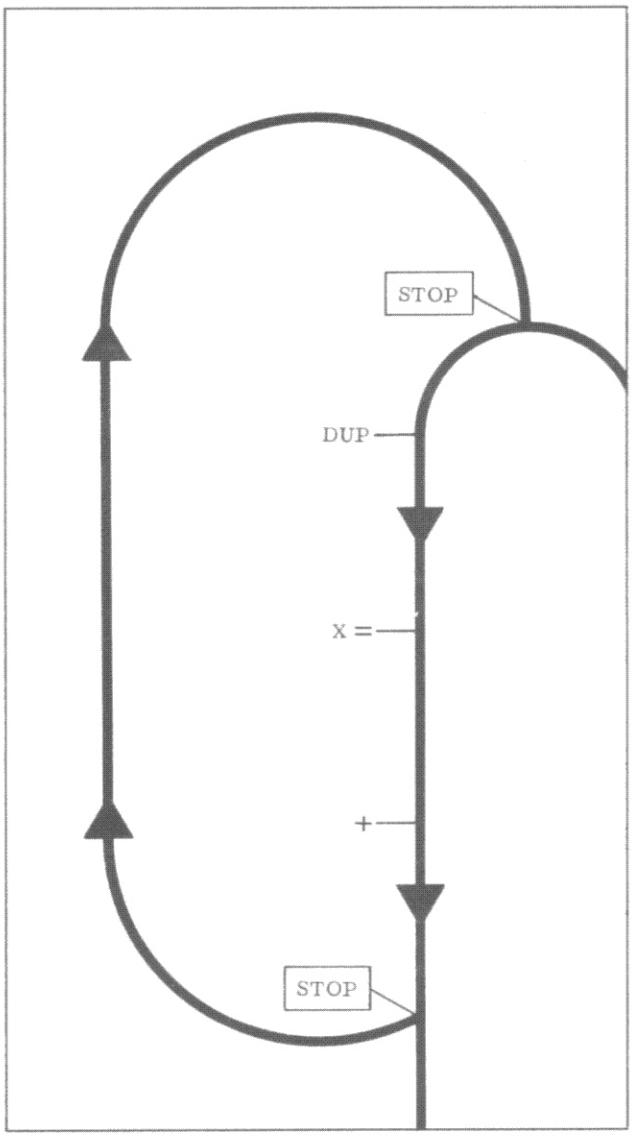
A sequence of instructions learned by the 1151 forms what may be referred to as a "loop". The diagram, below, illustrates this:



Notice that in the hypothetical sequence shown, we have two Stop Commands. Remember what was stated previously about the uses for a Stop Command:

1. To allow for manual calculations.
2. To allow for manual entries.
3. To separate a portion of a program from the remainder of the program.

The third reason for a Stop Command brings up the topic of "manual looping". A manual loop may be accomplished at any Stop Command in a program by depressing the PROGRAM RESET key. This will return, or "loop", the 1151 to the first step in the program sequence. The diagram below, illustrates this operation.



This ability to perform a manual loop allows us to re-use, indefinitely, the first portion of a program, avoiding the remaining instructions. The 1151 is the only desk-top programmable calculator that offers this program flexibility.

DECIMALIZATION AND OVERCAPACITY

The position of the Answer Decimal is immaterial when programming the 1151.

If the operator is using a learned program and inadvertently has set the Answer Decimal in the wrong position or incorrectly enters a variable, it is possible that an answer will exceed register capacity. If this happens, the automatic operation will stop, and the symbol "E" will print on the tape. The 1151 is now at the instruction in the program where overcapacity occurred.

The operator should depress PROGRAM RESET to return to the first step in the program, then check the decimal setting and the numbers entered for errors.

PROGRAMMING STEPS

The 1151 will learn any sequence of up to 30 steps of programmable instructions. Each programmable command counts as one step. Remember that if while the 1151 is learning a sequence:

1. A number is indexed and the FIRST NMBR/PRINT key depressed, a Stop Command (1 step) is recorded.
2. A number is indexed and a function key (X=, +, DUP, INTERCHANGE, etc.) depressed, a Stop Command (1 step) and the key function (1 step) are recorded.

Most common applications require less than 30 program steps. Therefore, it is not normally necessary to count the number of steps in a program.

BASIC PROGRAMMING

SIMPLE PROGRAMMING

Let's start with a simple program. The problem is this:

Formula: $A^2 + B^2 = C$

Data:

A	B
1	2
3	4
5	6

Problem: For each of the 3 sets of values for A & B, find the corresponding value of C.

Index	Touch
	LEARN
1	DUP
	X =
2	DUP
	X =
	+
	PRINT (First C value prints)
	PROGRAM RESET

We have now programmed the 1151 for this application. In this case (it is not always true, as you will see later on!) we were able to calculate our first answer while the 1151 was learning the formula. Let's look at the actual commands learned by the 1151:

Step	Command
1.	Stop (for entry of A value)
2.	Duplicate
3.	X =
4.	Stop (for entry of B value)
5.	Duplicate
6.	X =
7.	+
8.	Print (to print C value)

The 1151 is now at Step 1, waiting for the next A value. Let's get our second answer:

Index	Touch
3 (A)	AUTO
4 (B)	AUTO (Second C value prints)

The 1151 has accepted the new values for A and B, and printed the answer, C. The machine has automatically reset to Step 1 in the program sequence and stopped, waiting for the next value of A:

Index	Touch
5 (A)	AUTO
6 (B)	AUTO (Third C value prints)

Programming for such problems as this is obviously simple. We merely have to depress LEARN, calculate the problem once (just as you would manually, except that you must touch PRINT when an answer is in R1), and depress PROGRAM RESET. For a second set of values, merely enter each, in the proper sequence, with the AUTO key. The answer - or answers - automatically print, and the 1151 automatically resets, ready to accept another set of values.

STACK CONSTANT

The easiest way to hold a constant number would be to place the constant in the Memory Register, recalling it for use. In some applications, however, it is not practical to use the Memory Register for a constant number. For such occasions, let's describe a simple calculation where one term is a constant and we retain the constant in the stack.

Formula: $A^2 + B^2 = C$

Data: $B^2 = 4$

A
1
2
3

Problem: With $B^2 = 4$ a constant for the three problems, find the values for C corresponding to the given values of A.

Index	Touch
4 (B^2)	DUP LEARN
1	DUP X= + PRINT (First C value prints) INTERCHANGE DUP ($B^2 = 4$ is duplicated) PROGRAM RESET

The 1151 is now at Step 1 (Stop Command) in the program, waiting for a new value of A. The constant, $B^2 = 4$, is in R1 and R2. The last two program steps automatically regenerate our constant. Why did we use the INTERCHANGE key to place the constant number into R1 for duplication?

Manually, this could be done with the CLEAR key, to clear the value of C in R1, thus dropping the constant into R1. However, the CLEAR key is not a programmable function key!

Let's calculate the second C value:

Index	Touch
2	AUTO (Second C value prints)

Our program sequence has duplicated the constant and the 1151 is ready to accept the A value for the third calculation:

Index	Touch
3	AUTO (Third C value prints)

Now, let's look at a common application where this ability to hold a constant in the stack is important.

PAYROLL APPLICATION

To illustrate the use of the INTERCHANGE key to keep a constant in the stack, let's consider a payroll application. (Payroll calculations may be programmed in many ways, depending on the desires and requirements of the user. This is only one of many possible solutions.)

Data:

Regular hours - 40
Overtime hours - 6
Pay rate - $2.30/hr.

For this example, we assume that the overtime pay rate is 1.5 times the normal rate.

This is what our problem looks like:

40		x $2.30 = $ 92.00
9	(6 x 1.5)	x $2.30 = 20.70
		Total = $112.70

We are looking for dollars and cents answers, so we set the Answer Decimal on 2.

Index	Touch
40	TO MEMORY LEARN
2.30	DUP DUP FROM MEMORY X = PRINT (Regular pay prints) INTERCHANGE (Pay rate, 2.30, is now in R 1)
9	X = PRINT (Overtime pay prints) + PRINT (Total pay prints) INTERCHANGE (Pay rate, 2.30, is now in R 1) PROGRAM RESET

Our application is now programmed; we are at Step 1 in the program, the Stop Command for entry of the pay rate for the next employee.

Let's take a close look at our program. First, we have a constant, 40, which is kept in memory. Second, we are retaining a constant in the stack using the INTERCHANGE key. This second constant is the pay rate of $2.30.

There is a very big difference between the stack constant shown in the preceding section, and the stack constant in this application. The constants are retained in the same manner using the INTERCHANGE and DUP keys. The difference lies in the fact that in this payroll program, the operator has the option of using the constant (the pay rate) or using a new pay rate.

Let's say that the next employee is to be paid at the same pay rate.

Data:

Regular hours - 40
Overtime hours - 10
Pay rate - $2.30/hr.

We are at Step 1 in the program, the Stop Command at which the pay rate is to be entered. However, our program has retained the pay rate of $2.30 as a constant, now in R1:

Though a Stop Command allows for a manual entry, an entry need not be made to continue the program. If the AUTO key is depressed with no entry being made, the 1151 accepts the contents of R1 as the entry, and proceeds with the program.

Let's solve the second problem.

Index	Touch
	AUTO (Regular pay prints)
15 (10 x 1.5)	AUTO (Overtime, then total pay prints)

Again, the pay rate of $2.30 has been retained and is in R1, ready to be used again.

Let's say that our pay rate changes:

Data:

Regular hours - 40
Overtime hours - 4
Pay rate - $3.60/hr.

Index	Touch
3.60	AUTO (Regular pay prints)
6	AUTO (Overtime, than total pay prints)

Note that the stack constant is now 3.60! If the next employee's pay rate is $3.60/hr. we merely depress AUTO. If the next employee's rate is different, index the rate, then touch AUTO.

TWO TOTALS IN THE STACK

Quite frequently, we are faced with the problem of accumulating two (2) totals at the same time. Generally speaking, the easiest way to do this is to carry one total in the stack, the other in memory. Often, however, it is advantageous to be able to carry both totals in the stack, leaving the Memory Register available for accumulation of an additional total, or perhaps using it to retain a constant number.

Let's take a simple example of accumulating two totals in the stack:

Data:	x	y
	1	5
	2	6
	3	7
	4	8
Totals	10	26

Problem: We are to add both columns simultaneously.

Index	Touch
	CLEAR STACK
	LEARN
1 (First x value)	+
	INTERCHANGE
5 (First y value)	+
	INTERCHANGE
	PROGRAM RESET

Let's look at the program. Note that the last step places the number "1" (really, the subtotal to this point of the x values) into R1, and the number "5" (the subtotal to this point of the y values) in R2. There are no Print Commands in the program because we are not interested in seeing subtotals - only the final totals.

Let's enter our remaining x and y values:

Index	Touch
2 (x)	AUTO
6 (y)	AUTO
3 (x)	AUTO
7 (y)	AUTO
4 (x)	AUTO
8 (y)	AUTO

As you can see, each x value is added to the x subtotal. The contents of R1 and R2 are interchanged, placing the y subtotal in R1. Then, the corresponding y value is entered and added to the y subtotal. Finally, the contents of R1 and R2 are interchanged again, placing the x subtotal in R1, the y subtotal in R2.

After all x and y values have been entered, we need only touch the PRINT key. The resulting Stack Read-out shows the total of the x values (10) in R1 and the total of the y values (26) in R2.

Now, let's take a look at a practical application where this capability is put to use.

COST/SELL QUOTATIONS

In this application we are going to compute individual cost and sell, and the total cost and sell, figures for a quotation.

Data:

	Quantity	Cost/Unit	Price/Unit
Line (1)	10	$3.95	$4.85
Line (2)	12	8.91	9.95
Line (3)	23	1.26	2.49

To find the cost and sell figures for each line we must make two calculations:

Cost = Quantity x Cost/Unit
Sell = Quantity x Price/Unit

The quantity for each line is a constant for these two calculations. Therefore, we will use the Memory Register to hold it as a constant. When each of our two answers occurs in R1 we will cause it to print.

Additionally, we will sum the cost and sell figures in the stack, by the method shown in the preceding section. Set the Answer Decimal on 2.

Index	Touch
	CLEAR STACK
	LEARN
10 (Quantity)	TO MEMORY
3.95 (Cost/Unit)	FROM MEMORY
	X =
	PRINT (Cost, $39.50, Line (1) prints)
	+
	INTERCHANGE
4.85 (Price/Unit)	FROM MEMORY
	X =
	PRINT (Sell, $48.50, Line (1) prints)
	+
	INTERCHANGE
	PROGRAM RESET

This application differs from the example shown in the previous section. We are using the Memory Register to hold a constant used in the two multiplications in the program, and we are printing the products of these two multiplications.

Let's enter the figures for Line 2 of our quotation:

Index	Touch
12	AUTO
8.91	AUTO (Cost, $106.92, Line (2) prints)
9.95	AUTO (Sell, $119.40, Line (2) prints)

The 1151 has accumulated the cost and sell figures from Lines 1 and 2. Let's complete the problem:

Index	Touch
23	AUTO
1.26	AUTO (Cost, $28.98, Line (3) prints)
2.49	AUTO (Sell, $57.27, Line (3) prints)
	PRINT (Stack Read-out: total cost, $175.40 in R1, and total sell, $225.17, in R2, print)

MULTIPLE AND ITERATIVE PROGRAMS

LOOPING AT A STOP COMMAND

The 1151 has been designed so that two or more separate programs in one may be learned and used separately or in combination. Let's look at a fairly simple problem where this capability can be used.

Problem: $A = (1/2)(B^2 + C^2 + D^2 + E^2)^3$

B	C	D	E
.2	.3	.4	.5
.6	.7	.8	.9
1.0	1.1	1.2	1.3

There are two separate problems here:

(1) Sum the squares of the corresponding values of, B, C, D, and E.

(2) Raise that answer to the third power and divide by the number 2.

Let's see what the separate programs for these two problems would look like.

Problem (1)

Index	Touch
	CLEAR STACK
	LEARN
B	DUP
	x =
	+
	PROGRAM RESET

Problem (2) (Assume $B^2 + C^2 + D^2 + E^2$ is in R1)

Index	Touch
2	TO MEMORY
	LEARN
	DUP
	DUP
	x =
	x =
	FROM MEMORY
	÷ =
	PRINT
	PROGRAM RESET

To find the first value of A, we would have to use the first program four (4) times - for B=.2, C=.3, D=.4, and E=.5. We would then use the second program once to produce the value of A. How do we combine these two programs into one?

We have a constant, 2: let's put this in memory before we begin programming:

Index	Touch
2	TO MEMORY
	LEARN
0	DUP
	x =
	+
0	DUP
	DUP
	x =
	x =
	FROM MEMORY
	÷ =
	PRINT
	PROGRAM RESET

Let's look at the commands that the 1151 has learned:

Step	Command
1.	Stop
2.	Duplicate
3.	x =
4.	+
5.	Stop
6.	Duplicate
7.	Duplicate
8.	x =
9.	x =
10.	From Memory
11.	÷ =
12.	Print

There are two (2) Stop Commands in the program: Step 1, for entry of variable numbers, and Step 5, to separate the first and second portions of the program. As mentioned previously, at any Stop Command, we may manually loop to the first step in the program.

It was stated earlier that we cannot always calculate the answer to the first problem as the 1151 is learning the application. This is an example of such an application.

In general, usable answers are not generated while the 1151 is learning if one or more Stop Commands are used in a program for the purpose of manually looping.

Therefore, it is immaterial what numbers are used to create Stop Commands. We have chosen to use the number zero (0) in this case.

The 1151 is now at Step 1 of the program waiting for the first variable. Set the Answer Decimal on 2.

Index	Touch
.2(B)	AUTO

The 1151 is now at Step 5 of the program. There are three more variables to process (C=.3, D=.4, and E=.5) using the first portion of the program. Manual looping makes it possible:

Index	Touch
	PROGRAM RESET (Manual loop)
.3	AUTO
	PROGRAM RESET
.4	AUTO
	PROGRAM RESET
.5	AUTO

We are again at Step 5, and we have processed the four variables for the first value of A. We are now ready to use the second portion of the program: all that is required is to depress the AUTO key:

Index	Touch
	AUTO (Contents of R1 at Step 5, then, first A value, .07873, print.)

The answer to the first problem has printed, and the 1151 is again at Step 1, waiting for the next variable. The constant, 2, is still in memory, and the answer to the last problem is in R1. The first step is to clear R1; otherwise its contents will be added into the next calculation. Let's perform the second calculation.

Index	Touch
	CLEAR
.6	AUTO
	PROGRAM RESET
.7	AUTO
	PROGRAM RESET
.8	AUTO
	PROGRAM RESET
.9	AUTO
	AUTO (R 1 at Step 5, then, second A value, 6.08350, print.)

And for the third set of values:

Index	Touch
	CLEAR
1.0	AUTO
	PROGRAM RESET
1.1	AUTO
	PROGRAM RESET
1.2	AUTO
	PROGRAM RESET
1.3	AUTO
	AUTO (R 1 at Step 5, then, third A value, 76.13665, print.)

We have only used four variables (B, C, D, and E) in the first portion of the program. Obviously, we are not limited to that number: any number of variables may be processed, providing register capacity is not exceeded.

Let's take a look at a very practical example of manual looping, an invoicing application.

INVOICING

The common invoicing problem is a good illustration of manual looping to re-use a portion of program several times before using the remainder of the program.

Normally, an invoice is made up of a varying number of line extensions, discounts and taxes. The program for the problem must allow for any number of extensions. What answers should print depends on the needs of the user. If invoices are being checked, perhaps only the net amount should print. In this example we are assuming that invoices are being prepared. Therefore, we will cause the following answers to print:

1. Each extension
2. Invoice gross
3. Amount of discount
4. Subtotal after discount
5. Amount of tax
6. Invoice net

Problem:

Quantity		Price
10	@	$1.25 = $12.50
13	@	.98 = 12.74
26	@	2.46 = 63.96
		Gross = 89.20
		7% Discount = 6.24
		Subtotal = 82.96
		3% Tax = 2.49
		Net = $85.45

We need dollars and cents answers with automatic round off, so the Answer Decimal is placed on 2.

Here's the program:

Index	Touch
	LEARN
0	FIRST NUMBER
0	x =
	PRINT
	+
0	DUP
0	x =
	PRINT
	-
	PRINT
	DUP
0	x =
	PRINT
	+
	PRINT
	PROGRAM RESET

Let's look at the actual commands learned:

Stop	Command
1.	Stop (Quantity)
2.	Stop (Price)
3.	x =
4.	Print (Extension prints)
5.	+
6.	Stop (To separate portions of program)
7.	Duplicate
8.	Stop (Discount %)
9.	x =
10.	Print (Discount prints)
11.	-
12.	Print (Subtotal prints)
13.	Duplicate
14.	Stop (Tax %)
15.	x =
16.	Print (Tax prints)
17.	+
18.	Print (Net prints)

Look carefully at these program steps. Will this program automatically print all the answers that we require of it? There is no Print Command for the invoice gross - do we need a command for this?

When we have completed the last extension, the 1151 will be at Step 6 (Stop Command) of the program. At this point we want the invoice gross to print, and we want to continue with the program. The invoice total is in R1. If we depress AUTO to continue the program, the contents of R1 will automatically print. Having a separate command to print the invoice gross is therefore not required.

The 1151 is now at Step 1 in the program.

Let's clear the stack before we proceed - anything in R1 will be added to the first extension.

Index	Touch
	CLEAR STACK
10	AUTO
1.25	AUTO (Extension, $12.50 prints)
	PROGRAM RESET (Manual loop)
13	AUTO
.98	AUTO (Extension, $12.74, prints)
	PROGRAM RESET
26	AUTO
2.46	AUTO (Extension, $63.96, prints)
	AUTO (Gross, $89.24, prints)
.07	AUTO (Discount amount, $6.24, then subtotal, $82.96 print)
.03	AUTO (Tax amount, $2.49, then, net, $85.45, print)

The 1151 is at Step 1 in the program, ready for the next invoice. Let's explore some options available to us.

It's obvious that our program will handle any number of line extensions. What if the next invoice involved no discount off the gross and no taxes? The answer is simple: when you come to the Stop Command for entry of either of these values, enter a zero!

But what if there is, for example, a line discount, freight charges and credits? If these items were standard, we would probably write a different program. Let's consider that they are the exception to the rule, and see how they can be handled by the present program.

Remember that whenever the 1151 is at a Stop Command, we may perform manual calculations, then, continue with the program. This is exactly what we can do to handle the line discount, freight and credits.

Problem:

Quantity		Price
12	@	$11.12 less 10% = $120.12
8	@	5.03 = 40.24
		Gross = 160.36
		8% Discount = 12.83
		Subtotal = 147.53
		5% Tax = 7.38
		Subtotal = 154.91
		Freight = 25.05
		Credits = 10.95
		Net = $169.01

Index	Touch
	CLEAR STACK
12	AUTO
11.12	DUP
.1	x =
	- (Discounted price in R1)
	AUTO (Extension, $120.12, prints)
	PROGRAM RESET
8	AUTO
5.03	AUTO (Extension, $40.24, prints)
	AUTO (Gross prints)
.08	AUTO (Discount amount, $12.83, then, subtotal, $147.53, print)
.05	AUTO (Tax, $7.38, then, subtotal, $154.91 print)
25.05	+
10.95	-
	PRINT (Net, $169.01, prints)

After the 1151 printed the subtotal after tax, it reset to Step 1 in the program. Since this is a Stop Command and the subtotal was in R1, all we had to do was manually add in the freight charge, subtract the credits, and touch PRINT to obtain the invoice net. The 1151 is now ready for the next invoice!

ITERATIVE PROGRAMS

Essentially, an iterative program is one which must be repeated several times to

produce the answer or answers to one problem. Up to this point we have considered repetitive problems. The calculating steps were identical but the starting data (variables) changed from one problem to the next. In this section, we will consider repetitive use of a program sequence where each pass through the sequence generates the variables for the next pass.

Let's take a simple example.

COMPOUND INTEREST

To find the value at the end of one year of the principal (P) compound quarterly at an annual interest rate (i) we would use the following formula:

$$S = P\ (1 + \tfrac{i}{4})^4$$

The interest rate is constant, therefore, so is the value:

$$(1 + \tfrac{i}{4})$$

Let's assume:

P = $1,000
i = 5% per annum, compounded quarterly
or $S = (1{,}000)\ (1 \cdot \tfrac{05}{4})^4$

Here is what the program looks like to solve for the compounded value (S) of $1,000 at the end of one year. Move the Answer Decimal to 5.

Index	Touch
.05	FIRST NUMBER
4	÷ =
1	+ $(1 + \frac{i}{4})$
	TO MEMORY
1000	FIRST NUMBER
	LEARN
	FROM MEMORY
	x = (P) $(1 + \frac{i}{4})$
	FROM MEMORY
	x = (P) $(1 + \frac{i}{4})^2$
	FROM MEMORY
	x = (P) $(1 + \frac{i}{4})^3$
	FROM MEMORY
	x = (P) $(1 + \frac{i}{4})^4$
	PRINT (S, $1,050.94, prints)
	PROGRAM RESET

We have only indicated, above, that one answer printed - $1,050.94. However, when PROGRAM RESET was depressed, another number, $1,104.48, was calculated and printed. What is it?

Notice: There are no Stop Commands in the program. In the first chapter we stated that when the first step in the program is not a Stop Command for a manual entry, upon depression of the PROGRAM RESET key, the 1151 will proceed to the first Stop Command.

When no manual entries are made while the 1151 is learning a calculation, the 1151 will automatically place a Stop Command after the last step in the program. This automatic Stop Command does not count toward the total of 30 steps of programming instruction.

This means that when we depressed PROGRAM RESET an automatic Stop Command was inserted as the last step in the program. The 1151 cycled to Step 1 in the program: since this is not a Stop Command, it proceeded through the sequence to the automatic Stop Command.

If we were to use this program to calculate the compounded value at the end of one year of some other principal at the same interest rate, we could place the principal in R1 and depress PROGRAM RESET. This in fact, is what occurred. A new principal, $1,050.94, was in R1; when PROGRAM RESET was depressed, the 1151 accepted the new principal, then, calculated and

printed the compounded value, $1,104.48, for the second year!

The principal (P) for each successive year is a variable generated by the program. The starting principal for the third year is in R1. If we depress PROGRAM RESET, the 1151 will use this value to calculate and print the compounded value at the end of the third year.

You could proceed likewise for the fourth, fifth, years, etc. The process can be speeded up, however.

If no manual entries are made while the 1151 is learning, and the PROGRAM RESET key is held down, the 1151 will continuously cycle through the program sequence until the PROGRAM RESET key is released.

Let's see how this process applies for this application. The compounded value after two years of the starting principal, $1,000, has printed. Depress and hold down the PROGRAM RESET key. The values for each succeeding year will be automatically calculated and printed:

Year	Ending Balance
3	$1,160.75
4	1,219.88
5	1,282.03

Since there are no Stop Commands in the program, it may not be obvious how it is used to determine compounded values for a second problem. Actually, it is quite simple: using the new principal and interest rate, perform the manual calculations preceding depression of the LEARN key. Then, depress and hold down the PROGRAM RESET key.

Let's say that:

P = $2,500
i = 8% per annum, compounded quarterly

Index	Touch
.08	FIRST NUMBER
4	÷ =
1	+
	TO MEMORY
2500	FIRST NUMBER
	PROGRAM RESET (Depress and hold down)

As long as the PROGRAM RESET key is depressed, the 1151 will repeatedly cycle through the program sequence, printing the ending balance for each succeeding year:

Year	Ending Balance
1	$2,706.08
2	2,929.14
3	3,170.60
4	3,431.96
5	3,714.86

Let's look at another common iterative program.

SQUARE ROOT

The simplest programmed method for finding the square root of a number is known as Newton's Method of Approximations:

$$\sqrt{N} \simeq \left(\frac{1}{2}\right)\left(\frac{N}{a} + a\right)$$

The symbol "≃" is read, "is approximately equal to."

The number we are taking the square root of is "N": "a" is an approximation for the square root. When the values for "N" and "a" are substituted into the formula, we obtain a new, more accurate approximation. Inserting the new approximation into the formula, we obtain a still more accurate approximation for the square root of "N".

There are two constants in the formula: "N" and 2. The value of "a" is a variable generated by the formula.

Before we write the program for square root, let's touch briefly on a topic covered

later in more detail - number generation.

Here is an equation that always has the same answer:

$$\frac{x + x}{x} = \frac{2x}{x} = 2$$

Regardless what value we assign "x" the answer is always the number 2. Of what importance is this? What if we have the need for a constant, 2, in a program, but, because of the nature of the problem, have no way to carry that constant in the machine. We can very often solve that problem by using the above equation. Let's show that it does work using an arbitrary value for x:

Index	Touch
37 (x)	TO MEMORY FROM MEMORY DUP + (x + x) FROM MEMORY ÷=

The answer that printed was 2. Now let's use this ability to generate the number 2 in the program for square root. The program is similar to the one for compound interest in that we will make no manual entries while the 1151 is learning the calculation.

Problem:

$\sqrt{155} = 12.44989$

Set the Answer Decimal on 5.

Index	Touch
155(N) 12(a)	FIRST NUMBER FIRST NUMBER LEARN TO MEMORY DUP FROM MEMORY ÷= FROM MEMORY +
[	FROM MEMORY DUP + FROM MEMORY ÷= ÷=]
	PRINT (Second approximation, 12.45833, prints) PROGRAM RESET (Third approximation, 12.44990, prints)

Our program automatically retains 155 (N) as a constant. The constant number 2 is generated by the bracketed steps. Each new approximation calculated becomes the new variable, "a", for the next pass through the program. The last approximation calculated is in R1. Depress and hold down PROGRAM RESET:

Fourth Approximation = 12.44989
Fifth Approximation = 12.44989

The fourth and fifth approximations are identical. When this occurs, we have the answer. Regardless of the initial approximation we choose, the answers obtained through the program will eventually repeat to give us the square root. Let's see what happens when a bad approximation is used.

Problem:

$\sqrt{99.65} = 9.98248$

Just as in the compound interest program, for a new calculation, we must perform the manual calculations preceding depression of the LEARN key. This means we must place 99.65 in R2 and our first approximation in R1.

We will use an approximation just about as bad as possible. Let's use the number itself, as its own approximation.

Index	Touch
99.65	DUP PROGRAM RESET (Depress and hold down)

Approximations
#2 - 50.32500
#3 - 26.15256
#4 - 14.98144
#5 - 10.81650
#6 - 10.01463
#7 - 9.98253
#8 - 9.98248
#9 - 9.98248

It took longer for the answer to repeat, but the end result is the same regardless of what is chosen as the first approximation.

ITERATIVE PROGRAMS - GENERAL

Here are some more examples of iterative calculations:

SP 9496 - FINANCE

Page 5 - Amortization Schedule

SP 9497 - SCIENCE AND ENGINEERING

P. 4 - Cube Root
P. 5 - Sine of x
P. 6 - Cosine of x
P. 7 - Hyperbolic Sine of x
P. 8 - Hyperbolic Cosine of x
P. 9 - Exponential x
P. 10 - Natural Logarithms
P. 12 - Quadratic Equation
P. 15 - Conversion of Decimals to Common Fractions
P. 19 - Analysis of a Free Falling Body

The same basic principles as discussed in the preceding section apply to these programs. In each case, there is at least one variable generated by the program which is used for the next pass through the program sequence. Some of these programs (Sine, Cosine, etc.) do, however, have stop commands for manual entries. These stop commands are required due to the complex nature of the problems.

There is one, very critical, point regarding iterative programs that we have not touched upon:

In any iterative program, the register locations of the constants and variables at the first and last steps in the program MUST BE IDENTICAL.

Look closely at the programs for compound interest and square root. The above statement is true for these two programs, is it not?

The reason for this should be obvious. If the constants and variables for an application are not in the correct register, they will not be operated on properly by the programmed instructions. The result would be incorrect answers. The effect is the same as what would occur in an invoicing application if the percent discount were entered at the Stop Command for the price!

SPECIAL TECHNIQUES

GENERAL

There are many special techniques that can be used to advantage when writing programs for the 1151. We will not attempt to discuss all of them here. The ones shown can often be incorporated into standard programs to make them more efficient, or simply to tailor them to the specific needs of the user.

NUMBER GENERATION

We have already shown one example of number generation: the generation of the constant, 2, for our square root program. The same principle can be used to develop any whole number, within, of course, the program step capacity of the calculator:

$$\frac{nx}{x} = n$$

Where "n" is the whole number to be generated, and "x" is any arbitrary number.

Let's restate the above formula:

To develop any whole number, n, a constant for each pass through a program, add an arbitrary number, x, to itself n-times: then, divide the answer, (n) (x), by x.

There may be several ways to accomplish this, depending on the number to be generated, and the application in which it is used. Generally, it is best when x is in memory and, thus, may be recalled appropriately to satisfy the formula. There is only one requirement for x:

The number of decimal places in x should never exceed the Answer Decimal setting.

This could occur, for instance, if our problem requires the use of two (2) constants, one in memory, the other generated by the program using the memory constant. Our answers are to be printed at two decimal places, but the constant in the memory has five decimal places.

Let's write the sequence to develop the number 7; set the Answer Decimal on 5.

Index	Touch
2.13741 (x)	TO MEMORY LEARN FROM MEMORY DUP + (2) (x) DUP DUP + (4) (x) + (6) (x) FROM MEMORY + (7) (x) FROM MEMORY ÷ = PRINT (7 prints) PROGRAM RESET

This sequence takes any number that happens to be in memory and develops the number 7! To use this sequence in the program for an application, you would simply insert this list of instructions immediately preceding the point in the program where the number 7 is to be used. (You don't need the command PRINT, of course!) Isn't this exactly what we did in the case of the program for square root?

ROUND-OFF IN DIVISION

Round-off is automatic on the 1151 at any decimal setting (0-9) in addition, subtraction, and multiplication. Round-off in division can be provided by programming. The method shown here requires the use of the constant .1. The steps involved are identical, and the constant the same regardless of the Answer Decimal setting.

Let's .say that we have the problems:

$$149 \div 12 =$$
$$13.76 \div 1.2 =$$
$$1.27 \div .08 =$$

Our answers must be rounded-off at two (2) decimal places. Set the Answer Decimal on 2.

Index	Touch
.1	TO MEMORY (Memory constant)
	LEARN
149	FROM MEMORY
	÷=
12	÷=
	FROM MEMORY
	x =
	PRINT
	PROGRAM RESET

Division of the dividend, 149, by .1 moves the decimal place one position to the right so that when we divide by the divisor, 12, we obtain an extra significant digit (124.16). The answer obtained at this point is identical to the answer we would obtain if we divide 149 by 12 at decimal position 3, except that the decimal point is one position off.

To restore the decimal to its correct position, we then multiply 124.16 by .1. But we not only shift the decimal by this multiplication - we also get a rounded-off answer.

Let's complete the application:

Index	Touch
13.76	AUTO
1.2	AUTO (11.47 prints)
1.27	AUTO
.08	AUTO (15.88 prints)

Let's insert this sequence of instructions into an application:

Problem:

$$\frac{2^2 + 3^2 + 4^2 + 5^2}{36.536} = 1.480$$

Our answer is to be rounded-off at three decimal places: move the Answer Decimal to 3.

Index	Touch
.1	TO MEMORY
	LEARN
1	DUP
	x =
	+
[1	FROM MEMORY
	÷=
1	÷=
	FROM MEMORY
	x =
	PRINT]
	PROGRAM RESET

As you can see, the sequence within the brackets is the program for division round-off. Let's calculate the answer to the problem.

Index	Touch
	CLEAR STACK
2	AUTO
	PROGRAM RESET
3	AUTO
	PROGRAM RESET
4	AUTO
	PROGRAM RESET
5	AUTO
	AUTO (Dividend, 54.000, prints)
36.536	AUTO (1.478 prints)

The quotient, 1.478 is rounded-off at 3 decimal places as desired. And remember, once programmed, with .1 in memory, we can obtain rounded-off answers in division at any decimal setting. If the next problem requires a five (5) decimal place rounded-off answer, simply move the Answer Decimal to 5 and proceed!

COMPARISON TESTS

In iterative programs, such as square root, cube root, etc., each pass through the program produces an answer more accurate

than that obtained by the previous pass. When the last two answers are identical, we know that we have calculated the most accurate answer. This procedure requires that the operator compare the last two answers. Why not make the 1151 do this automatically!

Look back at the program for square root. The last step is PRINT. At this point, N is in R2, the latest approximation is in R1, and the previous approximation is in memory. Let's rewrite the program so that we subtract these two approximations, and print the difference. When the difference is zero, (0), we know that the last two approximations are identical. Move the Answer Decimal to 5.

Problem:

$$\sqrt{183} = 13.52774$$

Index	Touch
183	FIRST NUMBER
14	FIRST NUMBER
	LEARN
	TO MEMORY
	DUP
	FROM MEMORY
	÷ =
	FROM MEMORY
	+
	FROM MEMORY
	DUP
	+
	FROM MEMORY
	÷ =
	÷ =
	DUP
	FROM MEMORY
	-
	(PRINT (-.46429 prints)
	TO MEMORY
	PROGRAM RESET (Depress and hold down)

Second comparison: - .00796
Third comparison: - .00001
Fourth comparison: .00000

The difference between the last two approximations is zero, (0). Therefore we have the answer: it is in R1.

Index	Touch
	PRINT (13.52774 prints)

With this comparison technique, the operator does not have to compare the last two answers - the 1151 does that! All the operator has to do is watch the tape until a zero difference prints, then touch the PRINT key to print the final answer.

SEQUENCE SKIPPING

Applications arise that require a multi-part program which allows the operator the option of using or not using selected portions of a program. Up to this time, we have only considered the more frequent case where the operator repeatedly uses the first portion of a program, bypassing the remaining steps. What if the application requires that the operator repeatedly use the first portion, then repeatedly use a following portion?

Here is an example of such a problem:

Problem:

$$\sqrt{2^2 + 3^2 + 4^2} = 5.38516$$

Two problems are expressed here:

(1) Calculate the sum of the squares of 2, 3, 4.

(2) Calculate the square root of the answer.

Both of these problems require repeated use of their respective programs. We must be able to use the first portion (by manual looping) until we have accumulated the squares of 2, 3 and 4. We must then be able to enter an approximation for the square root of this value, enter, then repeatedly use the second portion of the program until we obtain the square root.

Here is the program:

Index	Touch
	LEARN
1	DUP
	x =
	+
1	DUP
	FROM MEMORY
	÷ =
	FROM MEMORY
	+
	FROM MEMORY
	DUP
	+
	FROM MEMORY
	÷ =
	÷ =
	PRINT
	TO MEMORY
	PROGRAM RESET

The first portion of the program is self-expanatory: each number to be squared is entered - we manually loop for the next number.

The second portion is a slightly modified program for square root. The second Stop Command has three functions:

1) Separate the first and second portions of the program.

2) After all values are processed by the first portion, allow us to print the answer.

3) Allow us to place the first approximation for the square root in memory.

At the second Stop Command, we will touch PRINT: the radicand in the formula will print. We will then place the first approximation in memory and depress AUTO to enter the second portion of the program.

Let's perform these steps:

Index	Touch
	CLEAR STACK
2	AUTO
	PROGRAM RESET
3	AUTO
	PROGRAM RESET
4	AUTO
	PRINT (29 prints)

Our approximation for the square root of 29 is 5.

Index	Touch
5	TO MEMORY AUTO (Contents of R1, 29, then second approximation, 5.4, print)

We have calculated the second approximation. The radicand, 29, is in R1, 5.4 is in memory. We are at Step 1 in the program. How can we skip the first portion of the program and use the second to give us the next approximation for the square root?

Index	Touch
0	AUTO

The 1151 has squared 0 (which is, of course, 0, again!) and added this to 29. We still have 29 in R1 and 5.4 in memory. This is exactly where these two values should be located at the first step in our square root program. All we need do is:

Index	Touch
	AUTO (29, then, third approximation, 5.38518, prints)

We must re-use the square root portion of the program until the approximations repeat.

Index	Touch
0	AUTO
	AUTO (29, then, fourth approximation, 5.38516, prints)
0	AUTO
	AUTO (29, then, fifth approximation, 5.38516 prints)

The approximations for the square root have repeated - we have our answer!

The procedure to skip a set of instructions is different for each application. As you can see, the instructions must be carefully planned. The numbers to be operated on by a portion of the program must be in the proper register locations at the first step of that portion.

Sequence skipping is not always possible. When it is, it allows us to get the most out of the 30 step 1151 program capacity!

SINGER
FRIDEN DIVISION

SP-9598

Printed in U.S.A.