TEXAS INSTRUMENTS
TI-66 PROGRAMMABLE

SOURCEBOOK

KEY INDEX

								[OFF] 1-4	[ON] 1-4
[A'] 3-12	[B'] 3-12	[C'] 3-12	[D'] 3-12	[E'] 3-12		[CSR] 4-44	[CMs] 2-7		
[A] 3-12	[B] 3-12	[C] 3-12	[D] 3-12	[E] 3-12	[EE] 2-9	[( ] 2-4	[ )] 2-4	[CE] 2-2	[CLR] 2-2
		[Ind] 4-73	[Exc] 2-7	[Prd] 2-8	[sin] 2-15	[cos] 2-15	[tan] 2-15	[Deg] 2-15	
[2nd] 2-6	[INV] 2-6	[STO] 2-7	[RCL] 2-7	[SUM] 2-8	[7] 2-2	[8] 2-2	[9] 2-2	[÷] 2-3	
[log] 2-14	[Intg] 4-26	[|x|] 4-26	[$\overline{x}$] 4-46	[$\Sigma+$] 4-44	[P►R] 2-17	[DMS-DD] 2-16	[π] 2-2	[Rad] 2-15	
[lnx] 2-14	[1/x] 2-12	[x^2] 2-12	[$\sqrt{x}$] 2-12	[y^x] 2-13	[4] 2-2	[5] 2-2	[6] 2-2	[×] 2-3	
[Part] 4-29	[Del] 3-6		[Nop] 3-53	[x = t] 4-67	[CP] 4-3	[Fix] 2-11	[Eng] 2-10	[Grad] 2-15	
[LRN] 4-58	[BST] 4-63	[SST] 4-63	[OP] 4-36	[x≠t] 4-40	[1] 2-2	[2] 2-2	[3] 2-2	[−] 2-3	[=] 2-3
[Dsz] 4-68	[Pause] 4-59	[IfF] 4-71	[StF] 4-71	[x≥t] 4-67	[Adv] 5-3	[Prt] 5-2	[List] 5-4	[Trace] 5-5	
[R/S] 4-58	[RST] 4-59	[GTO] 4-66	[SBR] 3-25	[LBL] 3-11	[0] 2-2	[.] 2-2	[+/−] 2-2	[+] 2-3	

Refer to *Appendix A* and the inside back cover for service and warranty information.

IMPORTANT

Record the serial number from the bottom of the unit and the purchase date in the space below. The serial number is identified by the words "SERIAL NO." on the bottom case. Always reference this information in any correspondence.

TI-66 PROGRAMMABLE		
Model No.	Serial No.	Purchase Date

TEXAS INSTRUMENTS
TI-66 PROGRAMMABLE
MANUAL

This book was developed by:

The staff of the Texas Instruments Instructional Communications Center

Kenneth E. Heichelheim
Robert E. Whitsitt, II

With contributions by:

Dick Ward
Linda Ferrio
Basil Melnyk
Danny Srader

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

Introduction

The Texas Instruments programmable 66 provides you with advanced scientific functions, large memory area, and user-friendly programming features. The calculator has arithmetic, logarithmic, trigonometric, statistical, polar to rectangular conversion, and other functions for use in calculations. The TI-66 can have a maximum of 512 program steps or 64 data memories with each memory convertible to 8 program steps.

When you are entering or reviewing a program, the calculator shows you readable abbreviations of the instructions, a significant improvement over key codes (used by earlier calculators).

The horizontal key layout and angled display give the calculator clean styling and functional grouping of the keys. The TI-66 is thin enough to slip into your pocket and weighs less than 5 ounces. Optionally, the PC-200 thermal printer connects to the TI-66 giving it printing and listing capabilities. Both units are compact and battery operated so you can take them anyplace you may need to solve mathematical problems. The long battery life eliminates the need for recharging. Your program and data, stored in the calculator, can be taken anywhere thanks to the calculator's constant memory feature.

The TI-66 uses the same set of instructions as the TI-59/58/58C family of calculators. *Appendix F* covers the differences between calculators.

Included with the calculator is a Quick Reference Guide and a complete sourcebook on operating and programming the calculator. This sourcebook has been specifically structured to start you programming right away. You'll see "hands on" how easy it really is to access the power of your TI Programmable calculator.

When you are looking at the instructions of a program, the display shows a mnemonic for each instruction. A mnemonic is a three (or fewer) character representation of an instruction displayed by the special alpha positions. An instruction's mnemonic is similar to the instruction's name. The mnemonic for pause is PAU and the mnemonic for run/stop is R/S. *Appendix E* provides a complete list of program mnemonics.

Chapters of This Book

Chapter 1 shows how easy it is to use and program your calculator.

Chapter 2 is a tour of the keys and functions of your calculator.

Chapter 3 is a guide to programming. Short lessons progressively present the concepts of programming the calculator.

Chapter 4 is a detailed and comprehensive analysis of functions and operations showing the effective limits of the calculator. (If you are already quite familiar with calculators and programming and just want all the facts and details right away, you may want to skip directly to *Chapter 4* and review your calculator in technical detail.)

Chapter 5 covers the use of the PC-200 printer with the TI-66.

Helpful and important information is also found in the appendices.

Power Up

New batteries were installed in your calculator at the factory. When the display becomes dim or erratic, the batteries need to be replaced. Just replace the batteries as instructed by the *Appendix A*.

Press the [ON] key. You will see a single zero and DEG in the display. Turning the calculator ON automatically establishes certain settings. These include setting angle units to degrees, removing scientific and engineering notation, assuming floating decimal point, setting the program pointer to the start, resetting all flags, clearing the subroutine return stack, clearing pending operations, and clearing the display. To check your calculator's display, press [8], decimal point [.] and the change sign [+/−] keys, then press [8] nine times to fill the display. An eight shows all segments of a standard display position. Note that the decimal point and minus sign progress to the left each time an eight is pressed. You can enter up to ten digits into your calculator at any one time for either positive or negative numbers. All digit entries made after the tenth are ignored. The minus sign always stays immediately to the left of any negative number in the display.

Whenever you exceed the limits of the calculator, Error is displayed. The error is cleared by pressing the clear key, [CLR].

Turning the calculator off (with the [OFF] key) and back on (with the [ON] key) removes the number in the display and any pending calculations. The *Constant Memory*™ feature retains numbers in user data memories and in the program memory. To conserve power, after about ten minutes of nonuse the calculator is automatically powered down through the *APD*™ *Automatic Power Down* feature. The effect is the same as if you had pressed the [OFF] key.

Take the time to explore the calculator. Investigate any feature you may be curious about. Check the manual's description of the feature. Try other features. This is one of the best ways to get to know just how much you can do with the calculator. The more you learn about its capabilities, the better it is able to serve your needs.

Types of Operations

There are two main uses of your calculator:

First, your machine is a high-powered manual calculator, ready to immediately handle the usual math chores as well as more intricate calculations with its advanced professional features. Second, the calculator can be programmed to give quick solutions to tedious formulas and repetitive problems.

Calculations From the Keyboard

Your advanced TI calculator is equipped with the AOS™ method of entering problems, one of the most straightforward entry methods yet devised. Problems are easily solved by entering them into the calculator simply and directly. For instance, to convert 100°C, 37°C, and −4°C to Fahrenheit, you multiply the Celsius reading by 9/5 and add 32. °F = °C × 9/5 + 32.

Press	Display
100 [×]	100
9 [÷]	900
5 [+]	180
32 [=]	212

You can repeat this sequence to find 37°C = 98.6°F and −4°C = 24.8°F. (More will be said about the AOS entry method and the calculating power it gives you later in the book.)

Writing Your Own Programs—An Example

Once you have determined a calculation sequence and you have several values to apply to that sequence, you can press the [LRN] (learn) key and teach the calculator the sequence. For the above example, press [LRN] then key in the following:

[×]
[9]
[÷]
[5]
[+]
[3] [2]
[=]
[R/S] (to stop and display answer)

Press [LRN] once more after the sequence. This tells the calculator to stop "learning" the keystrokes you enter. The calculator now remembers this sequence and can perform this series of operations on any number (in this case, any Celsius reading) that you may enter into the display.

1. Key in your Celsius value.
2. Press [RST] (reset). The program is at the step that you left it until the program pointer is moved. Reset immediately positions the program to ST, the point at the beginning of program memory (precedes step 000).
3. Press [R/S] (run/stop) to execute the program.

Press	Display
100 [RST] [R/S]	212
37 [RST] [R/S]	98.6
4 [+/−] [RST] [R/S]	24.8

This is all you need to do to convert any Celsius reading to a Fahrenheit equivalent. Writing your own program can be just as easy.

This ability to execute a program you have created is one of the most powerful aspects of your calculator. Once a program is stored and you have tested it to verify its accuracy, you can use it over and over again simply "at the touch of a key."

Printing Capabilities

Your calculator is compatible with the PC-200 printer. The printer can record the display value on paper whenever you tell it to. When solving problems directly from the keyboard, you can selectively print any or all desired intermediate results or provide a complete listing of a stored program. Print instructions encountered in the program cause automatic printing of the value in the display register. These printing features allow you to run a program while recording multiple answers. The trace option prints all steps performed and the corresponding numerical results.

Through use of the special control operations (OP codes) you can assemble and print any messages you need to identify segments of the listing or to place a title with a calculation sequence. Up to 16 characters can be printed per line, made up from a master set of 71 characters.

CHAPTER 2

Many users never fully access all the capabilities of their calculator, simply because they never take the time to see each key in action. This chapter can be covered in less than 30 minutes and provides information on the features and functions which make the TI-66 a high-powered scientific calculator. This tour will familiarize you with the main keyboard features so that as you move on into programming, you'll be able to take full advantage of the calculator.

Note: If you're already familiar with advanced calculators having the AOS entry method, you can skip this key tour chapter and get right into programming (*Chapter 3*). Refer to *Chapter 4* for an in-depth discussion of the calculator's features.

As you proceed through this tour, be sure your calculator is at hand. Demonstrate to yourself each key and feature as it's discussed. The best way to learn about your calculator is to use it!

Keyboard Basics

Clearing the Display—[CE], [CLR]

There are two procedures that allow you to clear the display register of your calculator, [CE] and [CLR].

[CE] Clear Entry—The clear entry key clears the last number you entered into the display (provided that a function or operation key has not been pressed). Use of this key does not affect calculations in progress. (So, if you accidentally hit 5 instead of 6 in the middle of an entry, just press [CE] and enter the complete correct number). The [CE] key will also clear an error condition.

[CLR] Clear—The clear key clears the contents of the display register and any calculations in progress. If an error condition exists when this key is pressed, it too is cleared.

Data Entry Keys—[0]-[9], [.], [+/−], [π]

Numbers are entered into the machine with the data entry keys [0] through [9], [.], and [+/−]. As you enter any number, the decimal point is assumed to be to the right of your entry until the decimal point key is pressed. The fractional part of the number is then keyed in, and the decimal point floats to the left with it. To change the sign of a number in the display just press the change sign key [+/−] once. (Pressing [+/−] again changes the sign back.)

Pressing [2nd] [π] places the first 10 digits of π in the display as 3.141592654. Thirteen digits are carried in the internal display register as 3.141592653590. [CE] does not remove this entry.

Basic Operation Keys—[+], [−], [×], [÷], [=]

Basic arithmetic is handled with the 5 basic operation keys [+], [−], [×], [÷], and [=]. Your calculator has a powerful feature called the AOS entry method. The AOS entry method automatically sorts out mixed operations in a problem for you. Even complicated problems can be entered simply and directly. (We'll say more about the AOS entry method below.)

When you press the [=] key, all pending operations (operations that AOS has delayed in order to perform higher ranked operations), are completed, and the result is displayed.

The AOS Entry Method

Mathematics never permits two different answers to the same series of operations. Because of this requirement, mathematicians have established a universal set of rules for calculations. For example, the problem:

$3 + 10 - 2 \times 14 \div 7 = ?$

has only one correct answer, 9.

You can key this problem directly, left to right, into your calculator and you'll get the correct result. The algebraic hierarchy of the calculator sorts the operations you enter, applies them in the correct order, and lets you see what it's doing along the way. Your calculator performs operations in the following universally accepted order:

1. Single argument function keys—act on the displayed number immediately—as soon as you press the key. (We'll talk more about each of these keys later in the "tour". They include all the keys for the trig and log functions and their inverses, as well as square, square root, reciprocal, integer and inverse integer, signum, absolute value, and conversions.)

2. Powers and Roots (y^x and $\sqrt[x]{y}$) are handled next (we'll discuss these further in this chapter.)

3. Multiplication and division are completed, followed by

4. Addition and subtraction.

This algebraic hierarchy applies to each set of parentheses.

Finally, the equals key completes all operations.

If you want to specify the order in which an expression is evaluated, you can do so with the parentheses keys, [( ] and [ )], which are discussed next. Parentheses receive the highest priority in mathematics and are treated that way by your calculator.

Parentheses Keys—[( ], [ )]

If you need to give any set of operations top priority, use parentheses. Parentheses give you a way to group numbers and operations. By putting a series of numbers and operations in parentheses you tell the calculator "Evaluate this part of the problem first, then use this result for the next part of the calculation." Within each set of parentheses, your calculator operates according to the rules of algebraic hierarchy. You should use the parentheses if you have any doubts about how the calculator will handle an expression.

You often see equations or expressions written with the parentheses to imply multiplication:

$(2 + 1)(3 + 2) = 15.$

Your calculator will not perform implied multiplication. You must key in the multiplication sign between the parentheses:

[( ] 2 [+] 1 [ )] [×] [( ] 3 [+] 2 [ )] [=] 15.

Here's an example using parentheses:

Evaluate: $\dfrac{8 \times (4 + 9) + 1}{(3 + 6 \div 2) \times 7}$

In problems of this type, you want the calculator to evaluate the entire numerator, then divide by the entire denominator. To ensure this, place an extra set of parentheses around the numerator and denominator as you key in the problem.

Press	Display	Comments
[CLR]	0	Clear any calculations in progress.
[( ] 8 [×] [( ] 4 [+] 9 [ )]	13	(4 + 9) is evaluated.
[+]	104	8 × (4 + 9) is evaluated.
1 [ )]	105	The value of the numerator.
[÷] [( ] [( ] 3 [+] 6 [÷] 2 [ )]	6	(3 + 6 ÷ 2) is evaluated.
[×] 7 [ )]	42	The value of the denominator.
[=]	2.5	The result.

Alternate Function Keys— [2nd], [INV]

Your calculator is equipped with numerous functions designed to save you time and increase the accuracy of your calculations. To allow you access to all of this power without loading the machine with keys, many of the calculator keys perform more than one function. The first function is printed on the key. To use the first function of a key, just press it. To use the second function (written above the key), just push the **[2nd]** key followed by the key below the function.

For example, to find the natural logarithm of a number, press **[lnx]**. To find the common logarithm of a number, press **[2nd] [log]**.

The inverse key **[INV]** also provides additional calculator functions without increasing the number of keys on the keyboard. When you press the **[INV]** key before a particular function or key, an alternate function of that key is accessed. The **[INV]** key works with many keys on your calculator to provide extra functions.

The **[2nd]** and **[INV]** keys allow over one hundred different keyboard operations to be performed even though the keyboard has only 49 keys. For use with specific keys, see *Alternate Function Keys* in *Chapter 4*.

Memory Keys— [CMs], [Part], [STO], [RCL], [Exc]

Each time you turn on your calculator there are 32 user data memories. Actually, the number of user data memories available versus the amount of program memory is variable. (See *Selection of Memory Size* in *Chapter 4* for details.) User data memories are locations in the calculator where you can store numbers you may need to use later. User data memories are also referred to as memories or data memories throughout this manual.

The number of data memories and the amount of program memory is governed by the partitioning. If partitioning is for no data memories, there are 512 program steps. If partitioning is for 64 data memories, there are no program steps. For each user data memory partitioned, the maximum program size is reduced by eight. Partition for XX data memories by pressing **[2nd] [Part]** XX.

Because there is usually more than one user data memory, you must indicate which memory you want to use by specifying its two-digit address XX. For example, [STO] 08.

For memories 0-9, a single digit can be used to address the memory provided a non-numeric key follows the address. For example, [STO] 8 [( ]. This is known as "short form addressing."

Pressing [2nd] [CMs] clears all data memories simultaneously (places a 0 in all memories). The [CE] and [CLR] keys do not affect what is in the memories.

[STO] XX (Store)—stores the number contained in the display register into memory XX (00-63, depending on the partitioning) without disturbing the contents of the display register. (Any number previously stored in memory XX is replaced.)

[RCL] XX (Recall)—This instruction simply brings the contents of memory XX to the display register. The contents of memory XX are not disturbed.

Example: Partition for eight data memories by pressing [2nd] [Part] 08. Store and recall 3.21.

Press	Display	Comments
3.21 [STO] 07	3.21	Store 3.21 in memory 7
[CLR]	0	Clear display
[RCL] 07	3.21	Recall contents of memory 7

[2nd] [Exc] XX (Memory Exchange)—The exchange sequence simply swaps the contents of memory XX with the contents of the display register. (The display register value is stored in memory XX while the number stored in memory is displayed.) This key allows you to make a quick check or use what is in memory without losing what's in the display register.

Memory Arithmetic Keys—[SUM], [Prd]

These key sequences let you operate on the numbers stored in memory without affecting pending operations or the value in the display register.

[SUM] XX (Memory Sum)—Adds the display register value into data memory XX. [INV] [SUM] XX subtracts the display register value from the data memory XX.

[2nd] [Prd] XX (Memory Product)—Multiplies the the display register value into data memory XX. [INV] [2nd] [Prd] XX divides the display register value into data memory XX.

Example: Calaulate the total cost of items of $28 and $6.60 with 5% sales tax. (Since cost + 5% = cost + .05 cost = 1.05 cost, a cost including 5% tax can be found by multiplying by 1.05.)

Press	Display	Comments
28 [STO] 01	28	Store 28 in memory 1
6.6 [SUM] 01	6.6	Add 6.6 to memory 1
1.05 [2nd] [Prd] 01	1.05	Multiply memory 1 by 1.05
[RCL] 01	36.33	Total Cost

Display Control

Standard Display

The display provides numerical information complete with negative sign and decimal point, indicates the angle units setting, and displays Error for an error condition. (A complete list of error conditions is found in Appendix B.) An entry can contain as many as 10 digits.

The terms *display* and *display register* are not synonymous. Display refers only to the digits you see in the calculator's display window. The display register is the internal register that retains numbers to 13 digits.

If a number is too large or too small to be handled by the standard format, the calculator automatically displays the number using scientific notation.

For example, when 400,000 and 2,000,000 are multiplied together you get 800,000,000,000, a number too large for the 10-digit display. So, it is displayed as 8 11 which means 8×10^{11}.

Scientific Notation Key— [EE]

In many applications, particularly in science and engineering, you may need to use very large or small numbers. Such numbers are easily handled using scientific notation. A number in standard form is just the number as it would be written with no exponent. A number in scientific notation is expressed as a number (mantissa) times ten raised to some power (exponent).

$$\text{Number} = \text{Mantissa} \times 10^{\text{Exponent}}$$

To enter a number in scientific notation:

1. Enter the mantissa using up to 7 digits. Then press [+/−] if the mantissa is negative.
2. Press [EE] (Enter Exponent). 00 appears at the right of the display.
3. Enter the power of 10. Then press [+/−] if the exponent is negative.

The number $-3.890145 \times 10^{-32}$ is displayed as −3.890145 −32.

In scientific notation, the exponent tells you how many positions the decimal point is from its position in standard form. A positive exponent tells you how many places the decimal point would be shifted to the right and a negative exponent tells you how many places the decimal point would be shifted to the left.

Example: $2.9979 \times 10^{11} = 299{,}790{,}000{,}000$
(Move decimal 11 places to the right and insert zeros as needed)

$1.6021 \times 10^{-9} = 0.0000000016021$
(Move decimal 9 places to the left and insert zeros as needed)

After you enter the scientific notation format it stays there until you remove it. If you press [INV] [EE], the calculator returns to standard display format as soon as the value in the display is within the range of the standard display. [CLR] removes scientific notation.

Engineering Notation Key—[Eng]

Engineering notation is a modified form of scientific notation. The power (exponent) is always adjusted to a multiple of three (10^{12}, 10^{-6}, etc.). As a result, the mantissa may have one, two, or three digits to the left of the decimal point. This feature allows the calculator to display results in units that are easily used by the scientist, engineer, or technician (such as 10^{-12} for picofarads, 10^{-3} for millimeters, 10^{3} for kilograms, or 10^{-6} for microseconds).

The display may be converted to engineering notation at any time by pressing [2nd] [Eng]. [INV] [2nd] [Eng] returns the display to standard display format. [CLR] does not remove engineering notation.

Example: Evaluate $8 \times 98 \times 30$ in Engineering Format.

Press	Display
[CLR] [2nd] [Eng]	0 00
8 [×] 98 [×]	784 00
30 [=]	23.52 03
[INV] [2nd] [Eng]	23520

Fix-Decimal Control—[Fix]

This convenient feature allows you to choose the number of decimal digits you'd like to appear in the display. Just press [2nd] [Fix], then press the desired number of decimal places (0 through 8). The calculator rounds all subsequent results to this number of decimal places for display only. However, the calculator retains its own internal accuracy of 13 digits. [INV] [2nd] [Fix] or [2nd] [Fix] 9 removes the fix-decimal format.

Example: 2 – 3 = 0.666666667

Press	Display
[CLR]	0
2 [–] 3 [=]	0.666666667
[2nd] [Fix] 6	0.666667
[2nd] [Fix] 2	0.67
[2nd] [Fix] 0	1.
[INV] [2nd] [Fix]	0.666666667

Algebraic Functions

Square, Square Root, Reciprocal Keys—[x²], [√x], [1/x]

These keys act immediately on the number in the display register without affecting other calculations in progress.

[x²] (Square)—Calculates the square of the value in the display register.

[√x] (Square Root)—Calculates the square root of the value in the display register.

[1/x] (Reciprocal)—Divides 1 by the value in the display register.

Here's an example putting them all together:
$\sqrt{4} \div (1/5)^2 = 50$

Press	Display	Comments
[CLR]	0	Clear any calculations in progress.
4 [√x]	2	$\sqrt{4}$
[÷] 5 [1/x]	0.2	1/5
[x²]	0.04	$(1/5)^2$
[=]	50	The result

Powers and Roots—[y^x]

The y^x key allows you to raise any positive number to a power. You can use [INV] [y^x] to find any root of a positive number.

For Powers (y^x)

1. Enter the number (y) you want raised to a power.
2. Press [y^x].
3. Enter the power (x).
4. Press [=] (or any operation key).

For Roots ($\sqrt[x]{y}$)

1. Enter the number (y) you want to find a root of.
2. Press [INV] [y^x].
3. Enter the root (x).
4. Press [=] (or any operation key).

Example: Calculate 2^6.

Press	Display
[CLR]	0
2 [y^x] 6 [=]	64

Example: Calculate $\sqrt[6]{64}$.

Press	Display
[CLR]	0
64 [INV] [y^x] 6 [=]	2

NOTE: You should only enter positive values for y. Error results from negative entries.

Logarithms—[lnx], [log]

These keys give you immediate access to the logarithms of any positive number without affecting calculations in progress.

[lnx] (Natural Logarithm)—Calculates the natural logarithm (base e = 2.718281828459) of the number in the display register. (Error is displayed if this number is negative or zero.) The antilogarithm of the natural log (e^x) is found by pressing [INV] [lnx]. Inverse natural log is valid for positive and negative numbers.

[2nd] [log] (Common Logarithm)—Calculates the common logarithm (base 10) of the display register value. (Again, the value in the display should be positive.) The antilogarithm of the common log (10^x) is found by pressing [INV] [2nd] [log]. Inverse common log is valid for positive and negative numbers.

Example: Calculate the natural logarithm of $(e^{2.7} + 10^{1.2})$.

Press	Display	Comments
[CLR]	0	Clear any calculations in progress.
[( ] 2.7 [INV] [lnx]	14.87973172	$e^{2.7}$ is evaluated.
[+] 1.2 [INV] [2nd]		
[log]	15.84893192	$10^{1.2}$ is evaluated.
[ )]	30.72866365	Pending addition is completed.
[lnx]	3.425195888	The result.

Angle Mode Keys—[Deg], [Rad], [Grad]

Your calculator is equipped to handle calculations that involve angles in degrees, radians, or grads. Your calculator always powers up in the degree mode. However, you may select any one of three common units for angular measure using the key sequences below.

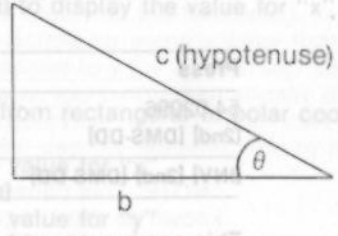
[2nd] [Deg] (Select Degree Mode)—In this mode all entered and calculated angles are measured in degrees, until another mode is selected. (There are 360° in a circle; a right angle equals 90°.)

[2nd] [Rad] (Select Radian Mode)—In this mode all angles are measured in radians. (There are 2π radians in a circle; a right angle equals $\pi/2$ radians.)

[2nd] [Grad] (Select Grad Mode)—In this mode all angles are measured in grads. (There are 400 grads in a circle; a right angle equals 100 grads.)

Trigonometric Keys—[sin], [cos], [tan]

These functions calculate the sine, cosine, and tangent of the angle held in the display register. The angle is measured in the units of the selected angle mode.



$$\cos\theta = \frac{b}{c} \qquad \sin\theta = \frac{a}{c} \qquad \tan\theta = \frac{a}{b}$$

where a, b, and c are the lengths of the sides.

The sequences **[INV] [2nd] [sin]**, **[INV] [2nd] [cos]**, and **[INV] [2nd] [tan]** calculate respectively the arcsine, arccosine, and arctangent. The resulting angles are displayed in units corresponding to the selected angle mode.

In the degree mode, all angles are interpreted in decimal format. (See Degree Format Conversions in the next section.)

Conversions

Degree Format Conversion— [DMS-DD]

There are two ways of representing an angle in degrees. One method is to use the degree/minute/second format, DDD.MMSSsss. DDD represents the whole angle, MM represents minutes, and SS denotes seconds. If greater accuracy is desired, fractional seconds may be entered in the sss position. Degrees are to the left of the decimal and minutes and seconds are to the right.

To convert from the degree/minute/second format to decimal degrees enter the angle into the display (DDD.MMSSsss) and press **[2nd] [DMS-DD]**. Pressing **[INV] [2nd] [DMS-DD]** converts decimal degrees to degrees, minutes and seconds.

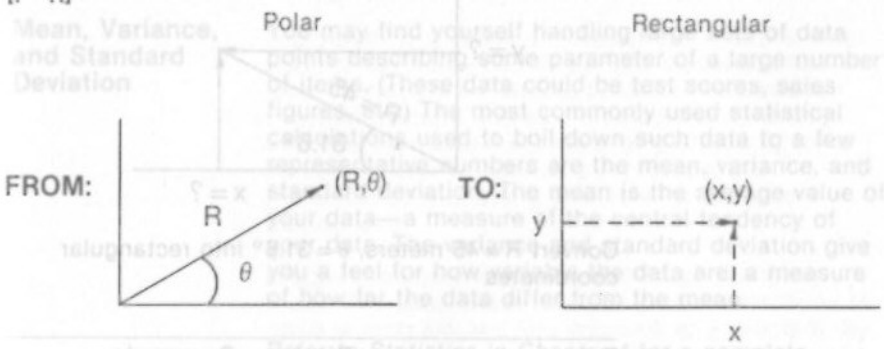
Two digits should always be entered for minutes and two for seconds as the calculator looks at the minutes and seconds part of the entry two digits at a time. Trailing zeros need not be entered.

Example: Convert 54°02'09.6" to its decimal equivalent and back.

Press	Display	Comments
54.02096		
[2nd] [DMS-DD]	54.036	DD.ddd
[INV] [2nd] [DMS-DD]	54.02096	DD.MMSSs

This same process can be used to convert hours, minutes and seconds to decimal hours and vice versa.

Polar/Rectangular Conversions—[P►R] The calculator makes it easy to convert between the polar and rectangular coordinate systems.



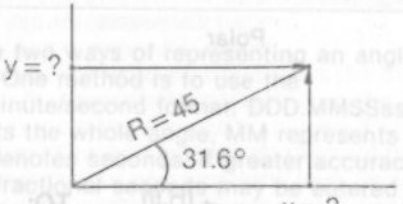
To convert from polar to rectangular coordinates:

1. Enter the value for "R"
2. Press [x⇄t]
3. Enter the value for "θ" (be sure angle mode is correct)
4. Press [2nd] [P►R] to display the value for "y"
5. Press [x⇄t] to display the value for "x"

To convert from rectangular to polar coordinates:

1. Enter the value for "x"
2. Press [x⇄t]
3. Enter the value for "y"
4. Press [INV] [2nd] [P►R] to display the value for "θ" in selected angle units
5. Press [x⇄t] to display the value for "R"

Example:



Convert R = 45 meters, $\theta = 31.6°$ into rectangular coordinates

Press	Display	Comments
[CLR] [2nd] [Deg]	0	Clear any calculations in progress and select degree mode.
45 [x⇄t]	0	*Place R in the t-register
31.6 [2nd] [P▸R]	23.57936577	Enter θ, convert to rectangular coordinates, and display y.
[x⇄t]	38.32771204	Display x. (y is now in the T-register)

*NOTE: This conversion uses a special register known as the t register accessed through the [x⇄t] (x exchange t) key. The special applications of this register are shown in the programming sections.

Statistical Functions

Mean, Variance, and Standard Deviation

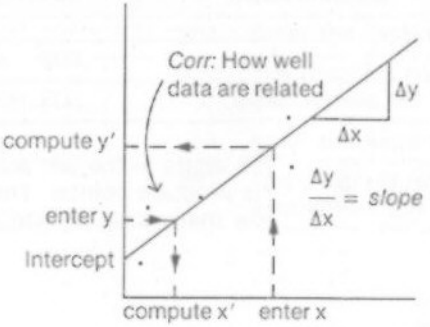
You may find yourself handling large sets of data points describing some parameter of a large number of items. (These data could be test scores, sales figures, etc.) The most commonly used statistical calculations used to boil down such data to a few representative numbers are the mean, variance, and standard deviation. The mean is the average value of your data—a measure of the central tendency of your data. The variance and standard deviation give you a feel for how variable the data are; a measure of how far the data differ from the mean.

Refer to *Statistics* in *Chapter 4* for a complete discussion of how to use these powerful functions.

Linear Regression

Linear regression deals with predicting future events.

In linear regression, data is usually expressed as pairs of variables that could be plotted on a graph. We usually label a pair of points like this with the letters x, y (x may be dollars in advertising while y is unit sales, or x may be a test score and y a performance record in the field, etc.). You want to make a prediction for some x value that you select: what will happen to y (or vice versa)? Your calculator can do this for you by mathematically drawing the "most representative line" through your data points. You may then use the resulting line to make predictions.



The use of these and other features is detailed in *Statistics* in *Chapter 4*.

Lesson 1—Storing a Program

A program is a series of instructions that you may wish to use numerous times. The calculator has an area of memory for storing program instructions. Program memory shares the calculator's memory with the user data memories. By partitioning for a certain number of user data memories, the remainder of memory is available for program instructions.

Program memory is accessed through the learn mode. The learn mode lets you store instructions to make a program and lets you look at a program. By numbering each program step, the calculator keeps track of the program position. Each time you enter an instruction, the step number advances. You can leave the learn mode to make keyboard calculations and come back to the learn mode at the same step. You must leave the learn mode to make keyboard calculations; otherwise, the keys you press are stored without being performed until the program is run.

In this book you will see the term "program pointer." The program pointer determines the step number to be displayed with each program step and causes program execution to follow the order of the program steps.

Try this simple program that adds two numbers.

first number + second number = answer

The program can be stated as follows.

Receive a value

Create pending addition operation, thereby remembering the current value

Receive a second value

Complete the operation

The Result

The program can be entered as shown here.

Press	Display		Comments
[2nd] [Part] 63	7 62		Set partitioning for 63 memories, 0-62 (this leaves 8 program steps, 0-7)
[2nd] [CP]			Clear program memory
[LRN]		ST	Enter learn mode
[+]	000	+	Add the number from the display
[R/S]	001	R/S	Stop to enter another number
[=]	002	=	Complete the calculation
[R/S]	003	R/S	Stop to show the answer

The digits in the left side of the display show you the program pointer. This allows you to keep track of how many program steps you have used.

Note that [R/S] has two different uses in this program. The first stop lets you enter a number at the right time. The second stop keeps execution from going beyond the end of the program. If you leave off the last [R/S], execution continues to step 004, 005, and so on until the end of program memory. The display remains blank until the program pointer reaches the end of program memory, and then Error is displayed.

Press	Display	Comments
[LRN]		Leave learn mode

The program pointer is still where you left it. Check the step number.

[LRN]	003 R/S	Enter learn mode
[LRN] [RST]		Leave learn mode and reset the program pointer to the start, ST

Before running a program, it is good practice to press [CLR] to ensure that there are no calculations pending.

Try using the program to add 227 and 34.

[CLR]		Clear possible pending calculations
227	227	Enter the first number
[R/S]	227	The program runs until it encounters the R/S
34	34	Enter the second number
[R/S]	261	The program runs until it encounters the next R/S

Try adding 107 and 107 by just entering 107 once.

[RST]		Reset the program pointer to the start
107	107	Enter the first number
[R/S]	107	The program stops with 107 in the display
[R/S]	107	The program runs until it encounters the next R/S

The answer is not what was expected. R/S does not affect the normal sequence of calculations. The sequence here is 107 + = which produces 107. For this program, a second number must be entered from the keyboard; the number is not reentered by R/S.

It may not seem like a very useful program, but some important fundamentals are shown here.

- A program is stored by keying in instructions while in the learn mode.
- The display register is useful for input and output.
- The program step number represents the program pointer.
- Program steps are numbered automatically.
- The program step number does not change until you take action to change it.
- Execution can be controlled by the use of R/S in a program and from the keyboard.
- R/S does not alter the calculation sequence.

Lesson 2—Editing

One drawback to entering numbers as the program stops for them is that you must keep track of program execution so you will enter the correct number at the proper time. It is better to assign each variable to a data memory before executing the program. That way you do not have to keep track of program execution. It is also better to have the calculator reset itself thereby relieving you of this task. This lesson shows how changes are made to a program.

The editing keys provide a means of changing a program and cannot be stored as program steps. While in the learn mode you can:

1. Display the instruction stored at any program location.
2. Delete instructions.
3. Insert instructions.

In order to understand the actions to take when editing, consider the four basic features of the editing process.

1. *You always see the instruction just written.* To show the instruction just written, every step must be automatically inserted after the previous step. If the instruction were not inserted, you would be writing over the next unseen program step.

2. *There is a point at the start called ST.* To have an automatic insert after each step, there must be a position before step 000. ST serves only to access step 000. It is not a step in program memory.

3. *There is no insert key.* Since every instruction is automatically inserted, no insert key is needed.

4. *The delete key has an automatic backstep.* Since the next instruction is inserted after the current step, a deleted instruction can be replaced most easily by including an automatic backstep. If you need to delete adjacent steps, it is easiest to delete the last one first and proceed in reverse order.

Keep in mind that every instruction is inserted—there is no "write over" action and that each time an instruction is written, all instructions after that point in the program shift one step later in the program. It takes less time to shift just a few instructions than it does to shift hundreds. You can shorten the calculator's response time to entering instructions by partitioning for just enough steps for your program before keying it in.

The program from Lesson 1, when changed, can be stated as follows.

Recall the number in memory 1

Create pending addition operation, thereby remembering the current value

Recall the number in memory 2

Complete the operation

Display the result; reset the program when the next run begins

The program can be entered by modifying the previous program as shown here.

Press	Display		Comments
[RST] [LRN]		ST	Enter learn mode
[RCL] 01	001	01	RCL is in step 000 and its memory address is in step 001
[SST]	002	+	Single step past existing program steps
[SST]	003	R/S	
[2nd] [Del]	002	+	Delete R/S and automatically backstep for the next entry
[RCL] 02	004	02	RCL is in step 003 and its memory address is in step 004
[SST]	005	=	Complete the calculation
[SST]	006	R/S	Stop to show the answer

[RST]	007 RST	When the program is run, it will reset itself. At this point, all program steps have been used. If [SST] is pressed or another instruction key is pressed, the calculator will leave the learn mode. To make room for more steps, simply repartition the memory outside the learn mode.
[LRN]		Leave the learn mode at step 7.

This was the program before editing.

000 +
001 R/S
002 =
003 R/S

This is the program after editing.

000 RCL
001 01
002 +
003 RCL
004 02
005 =
006 R/S
007 RST

Try using the program to add 227 and 34. If the program pointer is at step 7, the program will reset itself when run.

227 [STO] 01	227	Store the first number
34 [STO] 02	34	Store the second number
[R/S]	261	The program runs until it encounters R/S

Try using the program for 107 and 107.

107 [STO] 01	107	Store the first number
[STO] 02	107	The second number is already in the display
[R/S]	214	The program runs until it encounters R/S

The program used the numbers in memories 1 and 2. It added these numbers and stopped. This time, 107 is added to itself because it is in both memories. Pressing [R/S] executes a RST and repeats the program.

In some cases, leading zeros are not needed to access data memories 0-9. Short form addressing may be used whenever a nonnumeric keystroke immediately follows the memory address. In this program, [RCL] 01 [SST] could have been entered as [RCL] 1 [SST], since the calculator expects an address after RCL, the 1 is interpreted as 01 when followed by a nonnumeric key.

Note that RCL and 01 are two interdependent steps. Since there are many memories, RCL by itself would have no meaning so it requires an address. Several of the calculator's instructions require an additional part, forming an instruction group consisting of an instruction and its field. Depending on the instruction, the field is a memory address, a number of data memories, a label, a flag number, a number of display digits, an operation selection number, or a step number. If RCL is deleted from RCL 01, the calculator ignores 01 and regards it as a no-op.

[RST] [LRN]	ST	Enter learn mode
[SST]	000 RCL	Advance to the RCL
[2nd] [Del]	ST	Delete (backstep is automatic)
[LRN] [RST]		Leave the program.

Try using the program to add 5 and 4.

5 [STO] 01	5	Store the first number
4 [STO] 2	4	Store the second number (short form)
[R/S]	8	The 5 was not recalled but 4 was taken from the display and added to the number in memory 2.

Restore the program by inserting RCL.

[RST] [LRN]	ST	Enter learn mode
[RCL]	000 RCL	Insert RCL
[LRN] [RST]		Leave the program.

This program is useful only as an example since its function is simpler from the keyboard; however, several important concepts were presented.

- The learn mode operates with four important editing features.
- Instructions are automatically inserted.
- Some instructions use more than one step, the instruction and its field.
- Single step advances the program without altering the instructions of the program.
- Delete includes an automatic backstep.

Lesson 3—Labels

In each run of previous sample programs, you used [RST] and [R/S]. Since [RST] returns the program pointer to ST, you may have concluded that every program must start at the beginning of program memory. Labels provide easy access to any location within a program. You can start execution anywhere there is a label. A label occupies two steps: LBL followed by the label name. Here are some sequences from programs; in each is a label.

.	.	.
.	.	.
.	.	.
+	RCL	LBL
R/S	01	A
=	LBL	+
LBL	RCL	R/S
×	RCL	=
PAU	02	PAU
.	.	.
.	.	.
.	.	.

In the first sequence, label × is used. In the second sequence, label RCL is used. Note that a keystroke loses its original meaning when following LBL. The label serves only to mark a specific point in program memory and does not affect pending operations. A label should not be used to interrupt a sequence such as RCL 14 where more than one program location is involved in defining a single processing action.

In the third sequence, label A is used. This is called a user-defined label. There are ten user-defined labels available, A through E′ Press A from the keyboard and the calculator will find label A and begin running the program from that point. It eliminates the need to position the program pointer before execution.

Label x and label RCL are known as common labels. There are many of these labels on the calculator. Any key can be used as a label name except: [2nd], [SST], [BST], [LRN], [ON], [OFF], [Del], [Ind], [LBL], or a digit.

The difference between common labels and user-defined labels is that pressing a common label from the keyboard does not start program execution. If you have a program segment labeled x^2, for example, pressing [x^2] from the keyboard simply squares the displayed value. However, the keyboard sequence [SBR] [x^2] does cause the program to start running at label x^2. There are over 60 common labels to work with.

Change the program that adds two numbers to include user-defined labels.

Press	Display	Comments
[2nd] [Part] 61	23.60	Set partitioning for 61 memories, 0-60 (this leaves 24 program steps, 0-23)
[RST] [LRN]	ST	Enter learn mode
[LBL] A	001 A	Use A to make the first entry
[STO]	002 STO	
1	003 01	
[R/S]	004 R/S	Stop when the number is stored
[LBL] B	006 B	Use B to make the second entry
[STO]	007 STO	
2	008 02	
[R/S]	009 R/S	Stop when the number is stored
[LBL] C	011 C	Use C to make the calculation
[SST]	012 RCL	
[SST]	013 01	
[SST]	014 +	Add the two entries
[SST]	015 RCL	
[SST]	016 02	
[SST]	017 =	Complete the calculation
[SST]	018 R/S	Stop to show the answer
[LRN]		You left the program at step 18. The used-defined labels will take care of finding the right place to start.

Try using the program to add 227 and 34.

227	227	Enter the first number
[A]	227	The program stores the first number
34	34	Enter the second number
[B]	34	The program stores the second number
[C]	261	The program makes the calculation

Notice that the order of running A and B does not matter for this program. Try using the program to add 107 and 107.

107	107	Enter the first number
[B]	107	The program stores the first number
[A]	107	The second number is already in the display. The program stores the second number.
[C]	214	The program makes the calculation

The following comparison of the three addition programs we have tried provides an overall view of how the user-defined keys improve the usability of a program. Clearly, the third version is the easiest to use.

First Version	Second Version	Third Version
Press [RST]	Enter 227	Enter 227
Enter 227	Press [STO] 01	Press [A]
Press [R/S]	Enter 34	Enter 34
Enter 34	Press [STO] 02	Press [B]
Press [R/S]	Press [R/S]	Press [C]
Display 261	Display 261	Display 261

When processing transfers to a label, it preserves all pending operations, register contents, and settings and begins searching for the specified label at step 000. When it finds the label, it stops searching and changes the program pointer to the step number where the label was found. This process of transferring to a label takes a relatively short time, and does not noticeably delay running the program. If a label is at two places in a program, the second label will never be found. So do not use a label more than once in a program.

The important features of labels are:

- A label marks a position in a program.
- There are two types of labels: common and user-defined.
- LBL removes the original meaning from the instruction used as a common label.
- User-defined labels make a program more convenient.
- Each label should only be used once in a program.

Lesson 4—Transfers

A transfer moves the program pointer to a given destination. A transfer can be made from the keyboard (this technique is mainly used for getting to a certain point in a program so it can be edited) but transfers are mainly used in programs. When program execution reaches a transfer, it goes to the destination of the transfer and executes from there. The destination of a transfer can be a label or a step number.

The simplest transfer is GTO (pronounced "go to"). A segment that begins with a label and ends with a transfer back to the label is called a loop. Here is a program that uses a loop to count.

Press	Display	Comments
[2nd] [CP]		Clear program memory
[RST] [LRN]	ST	Enter learn mode
[LBL] [RST]	001 RST	Use this label for looping
[+]	002 +	Count by addition
[2nd] [Pause]	003 PAU	Display the number while the addition is pending
1	004 1	Increment by 1
[GTO] [RST]	006 RST	Loop to the label
[LRN]		Exit learn mode

Try having the program count starting with 10.

10	10	Enter starting value
[GTO] [RST]	10	Position the program pointer
[R/S]	10	Counting begins
	11	
	12	
	13	
	(etc.)	
[R/S] [CLR]	0	Stop the program from the keyboard (hold [R/S] until execution ceases) and clear any pending addition

This program can be streamlined. Since reset can be used as a transfer with destination of step 000 (RST works like GTO 000), the label can be left out.

[2nd] [CP]			Clear program memory
[LRN]		ST	Enter learn mode
[+]	000	+	Count by addition
[2nd] [Pause]	001	PAU	Display the number while the addition is pending
1	002	1	Increment by 1
[RST]	003	RST	Loop to step 000
[LRN]			Exit learn mode

Try having the program count starting with 10.

10	10	Enter starting value
[RST]	10	Position the program pointer
[R/S]	10	Counting begins
	11	
	12	
	13	
	(etc.)	
[R/S] [CLR]	0	Stop the program from the keyboard and clear any pending addition

These programs illustrate transfers and also rely on *AOS*™. A pending operation is completed when an operation of equal or lesser ranking in the hierarchy is executed. In this case, the pending + is completed the next time + is encountered.

You can have a step number as the destination of a transfer. This is called absolute addressing because the transfer always goes to the same step number address regardless of shifts due to editing. Remember that a step number is three digits. When the transfer uses absolute addressing, the destination will occupy two program steps. When you key in GTO 123, the following takes place.

[2nd] [CP]			Clear program memory
[RST] [LRN]		ST	Enter learn mode
[GTO]	000	GTO	Transfer instruction
1	002	01	Two steps after the instruction are set aside for the field
2	002	12	If the next keystroke is not a digit, the destination will be 012
3	002	23	The third digit completes the field. The first digit moved to the previous step.
[LRN] [RST] [LRN]		ST	Review the program
[SST]	000	GTO	Instruction
[SST]	001	01	First digit of destination
[SST]	002	23	Second and third digits of destination
[LRN]			Exit learn mode

Change GTO 123 to GTO 223.

[LRN]	002	23	Enter learn mode
[2nd] [Del]	001	01	Delete the last two digits
[2nd] [Del]	000	GTO	Delete the first digit
[2nd] [Del]		ST	Delete the instruction

[GTO]	000	GTO	Reenter the instruction so the calculator will merge the next three digits into two steps. The calculator would not merge properly if only the step that is wrong is changed.
2	002	02	Two steps after the instruction are set aside for the field
2	002	22	If the next keystroke is not a digit, the destination will be 022
3	002	23	The third digit completes the field. The first digit moved to the previous step.
[LRN] [RST] [LRN]		ST	Review the program
[SST]	000	GTO	Instruction
[SST]	001	02	First digit of destination
[SST]	002	23	Second and third digits of destination
[LRN]			Exit learn mode

Important points regarding transfers include:

- A transfer directs execution immediately to the chosen destination.
- The destination of a transfer can be a step number or a label.
- GTO is the simplest transfer.
- One effect of RST is that it transfers to 000.
- When absolute addressing is used, the destination occupies two steps.

Lesson 5—Planning a Program

A program can execute in sequence or it can transfer to distant locations in program memory. To make your programs understandable to other users, follow established ways of organizing programs. Your program might be many steps long. For someone else to understand it, you should break it into segments that each have a special job. If you know how a program should be organized, it will be easier for you to understand the action of someone else's program.

Generally, input begins a program. Start with those steps of the program that involve getting numbers into the calculator. Sometimes a program requires many inputs and the program takes quite a few steps before input is complete.

Often, a program relies on certain conditions to be in effect before execution reaches a certain point. These conditions may consist of clearing a memory, setting a flag, setting the number of times to loop, establishing the value in the t register, and other things which you will become aware of as your programming skills advance. Setting these conditions is usually done early in the program and is called initialization.

Next might be a section of computations. These would make up the main part of the program.

In the next example, user-defined labels are used to initialize, enter numbers, and to make the calculation. Here is a program to average numbers.

[2nd] [Part] 60		31 59	Set partitioning for 60 memories, 0-59 (this leaves 32 program steps, 0-31)
[RST] [LRN]		ST	Enter learn mode
[LBL] A	001	A	Use A to initialize memories
0 [STO] 01	004	01	
[STO] 02	006	02	
[R/S]	007	R/S	Stop when initialization is complete
[LBL] B	009	B	Use B to enter numbers
[SUM] 01	011	01	Memory 1 sums the numbers
[OP] 22	013	22	Memory 2 counts the entries
[R/S]	014	R/S	Stop when the number has been entered
[LBL] C	016	C	Use label C to make the calculation
[RCL] 01	018	01	
[÷]	019	/	Divide the sum by the number of entries
[RCL] 02	021	02	
[=]	022	=	Complete the calculation
[R/S]	023	R/S	Stop to show the answer
[LRN]			Exit learn mode

Try using the program to average 71, 81, 87, 84, and 92.

[A]	0	Initialize
71 [B]	71	Enter the numbers
81 [B]	81	
87 [B]	87	
84 [B]	84	
92 [B]	92	
[C]	83	Find the average

The program might end with the output but there might be some special segments of instructions after the output. This position at the end of the program is customarily reserved for subroutines. Subroutines are discussed in Lesson 6.

Here are some guidelines for developing a program on your own.

1. Define the problem clearly. Identify the formulas, variables, and desired results. What is known? What is to be determined? How are the known and the unknown related?

2. Develop a method of solution (an algorithm). Work out a preliminary sequence of operations that could be used to solve the problem.

3. It is often helpful to develop flow diagrams, drawings that depict the flow of execution. Here, you can picture interactions between various parts of the program. You may even discover ways to simplify the program structure after you see it drawn.

4. Assign labels to transfer points in the program. Decide which segments to access with user-defined keys. Decide if a segment will be more useful as a subroutine (see Lesson 6). Determine where the program should stop.

5. Assign user data memories for the various storage and "housekeeping" needs of the program.

6. Clear the program memory if you do not wish to use any of the program already entered. Enter the program. Do not use short-form addressing where a field is followed by a number. In the learn mode, single step through the program after it is entered to check for miskeyed instructions.

7. Test the program. Check out the program using test data.

8. Edit the program. Place the calculator in the learn mode and make the necessary corrections.

9. Retest the program. Repeat steps 7 and 8 as needed.

10. Record the program. Make a list of all program steps.

11. Document user instructions. Write down instructions describing how to use your program. Describe the restrictions and limitations of the program.

Important points regarding the planning of a program include:

- The order of most programs is as follows.

  Initialization/Input
  Computational sequences
  Output
  Subroutines

- The effort spent writing a program can be reduced through methodical development.

Lesson 6—Subroutines

Subroutines give you the capability to define a sequence of keystrokes that have a certain purpose. Once this sequence is made into a subroutine, it can be called in just two steps, an appreciable savings in program steps for a sequence that is needed repeatedly. You can label and reference subroutines from anywhere in your program. When you use a subroutine—it is said that you "call" it—you tell the calculator to temporarily go to a sequence of steps, run that sequence, and then to return to the point where the subroutine was called. When a subroutine is called, the calculator remembers the next execution step as the place to return to. The subroutine is executed until a return is encountered. RTN ([INV] [SBR] merges into RTN) directs execution back to the step stored when the subroutine was called.

It's good practice to write programs as subroutines so they can be used by other programs without modification. To do this, use [INV] [SBR] to halt program execution instead of [R/S]. Through the remainder of this manual, programs will use this technique when applicable.

The three stages of executing a subroutine are (1) calling it, (2) running it, and (3) returning to the point of call.

There are different ways of calling a subroutine:

- SBR to a label
- SBR to a step number
- user-defined key

In a program, a subroutine call stores the step number that will be used as the return address. If the subroutine begins with a common label, SBR followed by the label name is needed. If a subroutine begins at a certain step number, SBR followed by the absolute address is needed. If the subroutine begins with LBL A (a user-defined label), only A is needed to call the subroutine. As a rule, a subroutine which begins with a user-defined label can be called without the use of SBR. Here are some sequences from programs; in each is a subroutine call.

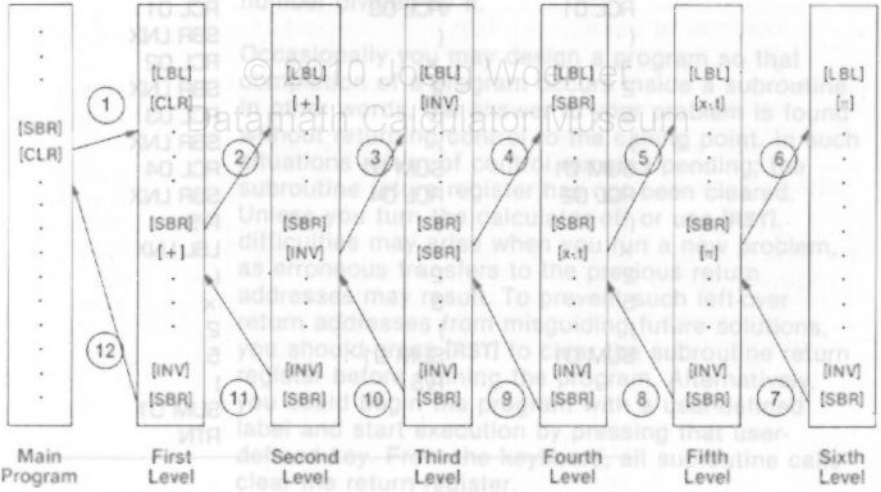
.	.	.
.	.	.
.	.	.
SBR	SBR	RCL
SBR	00	01
EXC	96	A
02	SBR	+
LBL	01	=
SBR	04	PAU
.	.	.
.	.	.
.	.	.

In the first sequence, subroutine SBR is called. Note that a keystroke, when used as a common label, looses its original meaning when preceded by SBR; that step only sends execution to the label. In the second sequence, the sequence beginning with step 96 is called and when execution returns, the sequence beginning with step 104 is called. In the third sequence, subroutine A is called. A user-defined label does not need SBR to call the subroutine.

Between the beginning of a subroutine and its return lie the steps that give the subroutine its function. This function might be to apply a formula to a number, place a number in memory, or to look up a number in memory. It is even possible to call another subroutine from a subroutine. The subroutine ends when execution reaches a RTN.

There is an internal memory for return addresses known as the return stack. It works much like placing numbered papers in a bin. When the stack is clear, the bin is empty. The paper with the first return address is placed in the bin first. The first number in is on the bottom of the pile—it will come out last. The most recent entry is on top. When a return occurs, the number on top is read, program execution returns to this step and the paper is discarded. The return stack is also called the return register.

The number of addresses in the return stack represents the level or number of subroutine in progress. Since the return stack can hold only six return addresses, a subroutine can call a subroutine that can also call a subroutine, etc., up to six times. This capability is shown graphically.



The return tells the calculator to transfer to the point in the program stored by the return stack. When a return occurs, processing checks for the most recently stored return address, transfers to that address, and clears that number from the return stack.

By providing a return for every subroutine call, the stack is cleared when no subroutines are being executed. If a RTN is encountered and the stack is empty, execution has no place to go, so it stops as if a R/S is there. Any time RST is used, the return stack is completely cleared.

If a subroutine is called from the keyboard, the return stack is cleared. When execution gets to the RTN at the end of the subroutine, it stops just as if a R/S were encountered.

The following example shows how a subroutine can be used to shorten a program.

Program Without Subroutines		Program With Subroutines
LBL A		LBL A
RCL 01	RCL 03	RCL 01
(	(	SBR LNX
×	×	RCL 02
2	2	SBR LNX
5	5	RCL 03
)	)	SBR LNX
SUM 01	SUM 01	RCL 04
RCL 02	RCL 04	SBR LNX
(	(	R/S
×	×	LBL LNX
2	2	(
5	5	×
)	)	2
SUM 01	SUM 01	5
	R/S	)
		SUM 01
		RTN

When attempting to reduce the number of program steps, look for sequences that appear more than once. If these sequences are long enough and do not contain RST, CLR, or =, replacing them with subroutines is worthwhile.

Avoid these instructions inside a subroutine: RST, CLR, and =. One function of RST is that it clears the return stack. When the stack is clear, no return will take place and any RTN encountered will stop execution. If you do need to transfer to location 000 (the primary function of RST), use GTO 000 or a label if there is one at 000. CLR clears all pending operations, those in the subroutine and in the main program. The equals instruction completes all pending operations including those of the main program and subroutines in progress. Instead of using =, enclose the operation in parentheses. An example of a subroutine built this way is

LBL SBR (/ PI) RTN

which takes the number in the display when the subroutine was called and replaces it with the number divided by π.

Occasionally you may design a program so that completion of a program occurs inside a subroutine. In other words, the answer to your problem is found without returning control to the calling point. In such situations return of control remains pending; the subroutine return register has not been cleared. Unless you turn the calculator off or use [RST], difficulties may arise when you run a new problem, as erroneous transfers to the previous return addresses may result. To prevent such left-over return addresses from misguiding future solutions, you should press [RST] to clear the subroutine return register before running the program. Alternatively, you could begin the program with a user-defined label and start execution by pressing that user-defined key. From the keyboard, all subroutine calls clear the return register.

The important features of subroutines are:

- The structure of a subroutine.
- The saving of steps that results when making a duplicated sequence into a subroutine.
- The three stages of executing a subroutine.
- How the return stack works.
- Things to avoid when writing subroutines.

Lesson 7—Decision Making

RST, GTO, and SBR transfer regardless of program conditions. There are also instructions that transfer only when a certain condition is met. Suppose you want a program to count as in Lesson 4, but you want it to stop once the count is high enough. The program can decide when to increment again and when to stop. Try this decision-making program.

Press	Display	Comments
[2nd] [CP]		Clear program memory
[LRN]	ST	Enter learn mode
[LBL] [E]	001 E	Label program
[( ]	002 (	Enclose in parentheses
[+]	003 +	Count by addition
1	004 1	Increment by 1
[ )]	005)	Complete addition
[2nd] [Pause]	006 PAU	Display the count
[INV] [2nd] [x≥t]	008 X≥T	Test if counting is less than t
[E]	009 E	Loop to label E until the count is high enough
[INV] [SBR]	010 RTN	End of sequence
[LRN]		Exit learn mode

Try having the program count from 6 to 10.

10 [x≥t]		Set the program to stop at 10 by placing 10 in the t register
6	6	Enter the starting value
[E]	7	Counting begins
	8	
	9	
	10	The program stops

A decision consists of two actions.

Action 1.	Test a condition. The result is either true or false
Action 2. (if true)	Transfer to given destination
Action 2. (if false)	Continue execution from this point if the condition is false

When the calculator is at action 1, it makes a test. The test can be a comparison of numbers, a special loop counter, or a flag. If true, a built-in go to sends execution to the destination. The destination can be a label or an absolute address (program step number). If false, execution skips this step and continues from that point in the program.

When the condition is a comparison of two numbers, the numbers must be where the calculator expects them: one in the display register (x) and the other in the t register (t). The t in t register stands for "test." For a comparison, the display number is tested against the number in the t register according to the current display format. It is necessary to store t in the t register and place x in the display register before making the comparison. A number is stored in the t register using X≷T. The comparison instructions are X≷T, X=T, INV X=T (not equal), X≥T, and INV X≥T (less than).

When the condition is a loop counter, a countdown of a data memory is made until its value is zero. It is necessary to initialize the correct user data memory before entering the loop. The loop counting instructions are DSZ and INV DSZ.

When the condition is a flag's status, it must be set to the intended status before the test is made. Setting a flag that has already been set, resetting a flag that has already been reset, and the testing of a flag have no effect on the status of the flag nor do they affect calculations. All flags can be reset at once with **[RST]** or **[2nd] [CP]**. The flag instructions are ST.F, INV ST.F, IF.F, and INV IF.F.

Note: When using a flag, check that the program can set *and reset* the flag to correctly indicate its intended meaning; otherwise, execution may get channeled to a path not originally intended for the conditions.

Taking into account the above preliminary tasks for making a decision, the process involves the following.

	Steps in program	Purpose
Preliminary		Set t and x, set the number of times to loop, or set the flag status.
Action 1.	Condition or INV condition	Test a condition. The result is either true or false.
Action 2. (if true)	Label or absolute address	Transfer to given destination.
Action 2. (if false)		Skip the transfer and continue execution starting with the next step

A simple way to state the execution of a decision is "transfer if true, skip if false."

Lesson 8—Examples of the three decision types

Comparisons

To illustrate comparisons, write a program to indicate a number is positive by returning a 1, negative by returning a −1, or zero by returning a 0. This is called the signum function.

Press	Display	Comments
[2nd] [CP]		Clear program memory
[RST] [LRN]	ST	Enter learn mode
[LBL] A	001 A	Label the program
[2nd] [CP]	002 CP	Set the t register to zero
[2nd] [x=t]	003 X=T	Check if X is equal to t
[=]	004 =	Transfer to label = if true
[INV] [2nd] [x>t]	006 X≥T	Check if X is less than t
[+/−]	007 +/−	Transfer to label +/− if true
1	008 1	Output for a positive number
[INV] [SBR]	009 RTN	End of sequence for positive numbers
[LBL] [+/−]	011 +/−	
1 [+/−]	013 +/−	Output for a negative number
[LBL] [=]	015 =	This label ends the sequence for 0 and negative numbers
[INV] [SBR]	016 RTN	
[LRN]		Exit learn mode

Try the program for 55, 0, and −299.

55 [A]	1	Positive
0 [A]	0	Zero
299 [+/−] [A]	−1	Negative

The DSZ Conditional Transfer

The DSZ instruction, *D*ecrement and *S*kip on *Z*ero, is ideal for counting the number of repetitions of a loop. When a sequence is needed a certain number of times, store the number in memory X (any of memories zero through nine) and place

DSZ X *destination*

at the end of the sequence. DSZ reduces the magnitude of memory X by 1 each time it is encountered and transfers to *destination* until the memory contains zero, at which point the transfer is skipped.

If memory X contains a noninteger, the content is decremented by ones until the last decrement when the fractional portion is subtracted.

Like the other transfer instructions, DSZ can be used from the keyboard but is usually used in a program.

To illustrate DSZ, write a program to count by ones from 1 to an entered value.

[RST] [LRN]			ST	Enter learn mode
[LBL] A		001	A	Label the counting program
[STO] 09		003	09	Memory for countdown
[STO] 10		005	10	Memory for constant

[LBL] [2nd] [Dsz]	007	DSZ	The loop starts here
[( ]	008	(	
[RCL] 10	010	10	Calculate the difference between the constant and the countdown and add 1
[+] 1 [−]	013	−	
[RCL] 09	015	09	
[ )]	016	)	
[2nd] [Pause]	017	PAU	Display the present count
[2nd] [DSZ] 9	019	09	Decrement memory 9
[2nd] [DSZ]	020	DSZ	Transfer to label DSZ if memory 9 is not yet zero
[INV] [SBR]	021	RTN	End the sequence when the countdown is complete
[LRN]			Exit learn mode

Try running the program to count to six.

[CLR]		Clear the display and pending operations
6 [A]	1	Counting begins
	2	
	3	
	4	
	5	
	6	

Notice in step 019, the field for DSZ is 09 but it was entered as 9. DSZ works only for memories 0 through 9. It accepts only a ones digit in the address field even though two digits are displayed.

Flags in a Program

There are 10 individual flags, numbered 0-9. Some flags are internally programmed to perform special functions as follows.

Flags 0-6	General purpose flags.
Flag 7	[OP] 18 sets flag 7 if no error condition exists. [OP] 19 sets flag 7 if an error condition does exist.
Flag 8	Setting flag 8 causes the calculator to stop a program if an error occurs while a program is running.
Flag 9	If you are using your calculator with the optional printer, you may control the trace mode of the printer with flag 9. If flag 9 is set, the printer is placed in the trace mode and calculated results are printed after each function or operation. If flag 9 is reset, then results are printed only by a print instruction. Flag 9 may be used as a general purpose flag if you are not using the optional printer.

Flags have numerous uses, three of which are listed below.

- Controlling program options manually from the keyboard before running a program
- Program conditions set a flag for later testing
- Keeping track of execution history—which path through the program has taken to the present point?
- Setting a non-numeric condition in a program

To illustrate flags, write a program to add numbers to memory 1 or memory 2 depending on the setting of a flag. This program could be used to total sales made in the morning separately from those made in the afternoon without the salesperson being concerned with the separation of totals. These totals could help determine how many salespeople to use in the morning and afternoon. The salesperson would just enter the sale and press [A]. Simply have the store manager press [D] at the start of the day, and [E] at noon.

[RST] [LRN]		ST	Enter learn mode
[LBL] A	001	A	Label the sales program
[2nd] [IfF] 0	003	00	Flag zero, if set, totals morning sales
[+]	004	+	Transfer to LBL + for the morning
[SUM] 00	006	00	Afternoon total in memory 00
[INV] [SBR]	007	RTN	
[LBL] [+]	009	+	
[SUM] 01	011	01	Morning total in memory 01
[INV] [SBR]	012	RTN	
[LBL] D	014	D	
[2nd] [StF] 0	016	00	Set flag for morning
[INV] [SBR]	017	RTN	
[LBL] E	019	E	
[INV] [2nd] [StF] 0	022	00	Reset flag for afternoon
[INV] [SBR]	023	RTN	
[LRN]			Exit learn mode

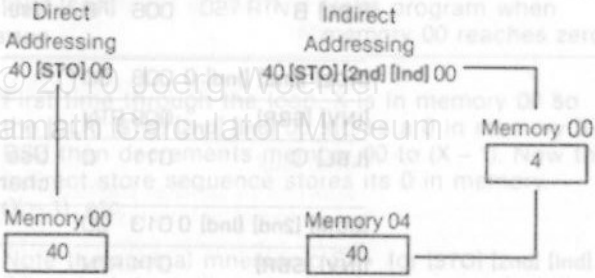
Press [D] to begin the morning and press [E] to begin the afternoon. Enter the sales by pressing [A].

0 [STO] 01	0	Clear memories 01 and 00
[STO] 00	0	
[D]	0	Morning begins
29.95 [A]	29.95	
14.69 [A]	14.69	
[E]	14.69	Afternoon begins
59.48 [A]	59.48	
19.95 [A]	19.95	
[RCL] 01	44.64	Morning sales total
[RCL] 00	79.43	Afternoon sales total

Lesson 9—Indirect Addressing

Additional capabilities can be added to data memory operations, transfer sequences, and special control addressing through use of the indirect instruction, **[2nd] [Ind]**. The basic concept is that you go to some user data memory (the indirect address), not to find the information you need, but for where to find the information. It is sometimes much easier to obtain information indirectly like this. Instructions are used indirectly by placing **[2nd] [Ind]** and then the indirect address after the instruction. At this indirect address is found the information that is actually needed.

A sample direct and indirect store instruction is illustrated below.



Here is an example that uses indirect functions to keep track of the balances in several different accounts.

Press	Display	Comments
[2nd] [Part] 60	31.59	Set partitioning
[2nd] [CP]	31.59	Clear any previous program
[LRN]	ST	Enter learn mode
[LBL] A	001 A	Use label A to enter the account number
[STO] 0	003 00	Store the account number in memory zero
[INV] [SBR]	004 RTN	
[LBL] B	006 B	Use label B to check the current balance
[RCL] [2nd] [Ind] 0	008 00	
[INV] [SBR]	009 RTN	
[LBL] C	011 C	Use label C to enter a change in the balance
[SUM] [2nd] [Ind] 0	013 00	
[INV] [SBR]	014 RTN	

Of course indirect could be used in a non-repetitive application, but it is usually more efficient to apply indirect addressing where an operation is needed repeatedly but for varying addresses. Continue entering the following segment in this program.

This program segment clears memory 1 through X where you can vary X.

Key Sequence	Program Step	Comments
[LBL]	015 LBL	Use label D to enter X
[D]	016 D	

[STO]	017	STO	Store X in memory 00
0	018	00	
[LBL]	019	LBL	
[E]	020	E	
[CLR]	021	CLR	
[STO] [2nd] [Ind]	022	ST*	Zero is to be stored where memory 00 says to store
0	023	00	
[2nd] [Dsz]	024	DSZ	DSZ loop on memory 00
0	025	00	
[E]	026	E	Go to E if memory 00 not zero
[INV] [SBR]	027	RTN	Halts program when memory 00 reaches zero

First time through the loop. X is in memory 00 so the [CLR] [STO] [2nd] [Ind] 00 stores a 0 in memory X. DSZ then decrements memory 00 to (X – 1). Now the indirect store sequence stores its 0 in memory (X – 1), etc.

Note the special mnemonic ST* for [STO] [2nd] [Ind]. Several of the indirect instructions are merged like this to save program space. For a complete list, see page 4-73.

You can use 1 through the highest memory partitioned for account numbers. Remember to establish the starting balances by clearing the memories and entering the starting balance as each account's first deposit. A positive amount is considered a deposit and a negative amount is considered a withdrawal.

Indirect addressing is provided for all of the calculator's functions that have a field (except LBL). When a series of addresses can be easily calculated, indirect addressing saves program steps especially when taking advantage of a decrement instruction.

Indirect addressing can be used on either or both fields of an instruction that has two parameters. Indirect flag control is accomplished by placing the number of the flag in a user data memory. For example, storing 6 in memory 12 and completing the sequence **[2nd] [StF] [2nd] [Ind]** 12 sets flag 6 while **[2nd] [IfF] [2nd] [Ind]** 12 **[2nd] [Ind]** 12 branches to step 006 if flag 6 is set.

See *Indirect Addressing* in *Chapter 4* for a complete list of indirect instructions.

Important points regarding indirect addressing include:

- The form of an operation using indirect addressing is:
  operation key
  [2nd] [Ind]
  address of pointer memory.
  (two-parameter functions only)—**[2nd] [Ind]** address of second parameter pointer memory.
- The pointer memory indicates the actual address that will be used.

Lesson 10—Program Optimization

There are many methods of combining separate program parts to save space. For instance, if a subroutine call occurs as the last operation of a routine, you may place the subroutine in line with the first and eliminate the subroutine call.

A program like this	can look like this
.	.
.	.
.	.
LBL	LBL
E	E
.	.
.	.
.	.
SBR	LBL
STO	STO
R/S	.
.	.
.	.
LBL	RTN
STO	
.	
.	
.	
RTN	

Not only is a savings of several steps realized, but one level of the subroutine return register has been freed. [INV] [SBR] now acts like a [R/S], because the subroutine return register is clear.

As another illustration, consider the two sequences shown below:

Workable Segment	Efficient Segment
.	.
.	.
X=T	X=T
D	D
1	LBL
STO	D
04	1
RTN	STO
LBL	04
D	RTN
1	
STO	
04	
RTN	

The purpose here is to store a .1 or a 1 depending upon the results of the test. Both of these routines perform the same function; however, the second is four steps shorter than the first. The second segment is organized so as to eliminate the first segment's need to repeat steps.

In addition to the various techniques of combining separate routines there are also numerous programming tricks that you may find valuable. In the next example the programmer desires to use only the value rounded to two decimal places of the number displayed in his calculations. Simply placing the calculator in fix-decimal does not work because most calculations continue to use the 13 digit display register value.

Workable Segment	Efficient Segment
.	.
.	.
.	.
(	FIX
(	02
X	EE
1	INV
0	EE
0	FIX
+	09
.	.
5	.
)	.
INT	.
/	.
1	.
0	.
0	.
)	.
.	.
.	.

The purpose and method of the routine on the left are fairly straightforward. The reasoning behind the second sequence is more efficient but also more obscure. Since the EE instruction operates only on the displayed digits, this instruction discards the unwanted digits after placing the display in fix-decimal. The routine then removes the scientific notation format and continues using only the rounded value.

The following routines demonstrate three methods of performing the same operation: adding 10,000 to the display register.

.	.	.
.	.	.
.	.	.
(	(	(
+	+	+
1	1	4
0	EE	INV
0	4	LOG
0	)	)
0	.	.
)	.	.
.	.	.
.	.	.

Both the second and the third routine require the same number of program locations. The second method, however, is advantageous only when you wish to leave the display in scientific notation.

As you become more acquainted with the capabilites of your calculator, you will undoubtedly discover short cuts that fit your needs. Be sure to record these sequences for future use as they will lessen the programming task. Until then, you may use the many step-saving features already built into your calculator in optimizing programs. These features include functions such as the memory operations [SUM] and [2nd] [Prd], indirect instructions, and the many special control operations.

Sometimes you may be attempting to program in too straightforward a manner. You can often come up with a different solution method and save program steps compared to the first solution attempt. This is illustrated in this next example.

Service Charge Program

A manager of a bank needs a fast and easy method of determining the monthly service charge for many customers.

The service charge for each account is calculated as follows:

$0.10 per check for the first five checks (1-5).
$0.09 per check for the next five (6-10).
$0.08 per check for the next five (11-15).
$0.07 per check for each check over 15.

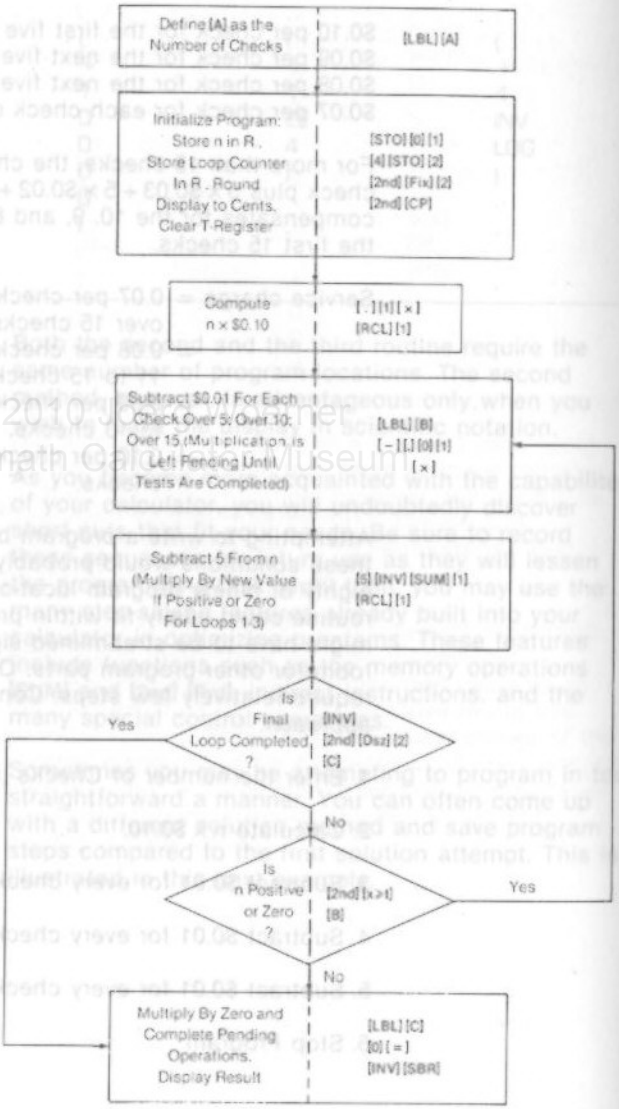
For more than 15 checks, the charge is 7 cents per check plus $5 \times \$0.03 + 5 \times \$0.02 + 5 \times \$0.01$. This compensates for the 10, 9, and 8 cent charges for the first 15 checks.

Service charge = 0.07 per check + 0.30 if there are over 15 checks.
= 0.08 per check + 0.15 if there are 11 to 15 checks.
= 0.09 per check + 0.05 if there are 6 to 10 checks.
= 0.10 per check if there are 1 to 5 checks.

Attempting to write a program based directly on these conditions would probably require at least eighty or ninety program locations. Although such a routine could easily fit within program memory, it might have to be streamlined significantly to allow room for other program parts. One solution would require relatively few steps. Consider the following approach.

1. Enter the number of Checks (n).
2. Calculate $n \times \$0.10$.
3. Subtract $0.01 for every check except the first 5.
4. Subtract $0.01 for every check except the first 10.
5. Subtract $0.01 for every check except the first 15.
6. Stop Program

Instead of testing the number to see which category it fit into and then applying a formula, this approach applies an equal charge to all the checks and then discounts the charge for the checks above each cutoff point. Examine the logic here for a moment.



The program is fairly straightforward until location 022 where the multiplication in step 021 is left pending while an adjustment is made to n and tests are completed. The loop is used to reduce the charge on each check over 5 to $0.09; over 10 to $0.08; over 15 to $0.07. The [2nd] [Dsz] instruction asks which loop is in progress. For loops 1 through 3, the value of n is tested; if it is negative, zero is placed in the display to complete the pending multiplication and the program is terminated upon computing the total service charge.

If the fourth loop is reached, the pending multiplication is always completed with zero, otherwise the charge on each check over 20 would be reduced to $0.06. The program then determines the total service charge and halts the program. This last loop is not necessary for computation; however, its elimination would require the use of additional program instructions and the idea is to minimize the size of the routine.

Service Charge Program Listing

000 LBL	020 1
001 A	021 X
002 STO	022 5
003 01	023 INV
004 4	024 SUM
005 STO	025 01
006 02	026 RCL
007 FIX	027 01
008 02	028 INV
009 CP	029 DSZ
010	030 02
011 1	031 C
012 X	032 X≥T
013 RCL	033 B
014 01	034 LBL
015 LBL	035 C
016 B	036 0
017 –	037 =
018	038 RTN
019 0	

To run the program, simply key in some number of checks and press [A]. For instance, 1 check costs $0.10, 6 checks cost $0.59 and 63 checks cost $4.71.

Only two approaches have been considered for this service charge problem. Realizing that there are many ways to program the solution to a problem, these two extremes show just how different programming techniques can be. Naturally, there are trade-offs. In this instance the second method requires less than half the program space needed for the first method; however, the first method demands less time for the program to run. Regardless of the approach you take to programming, the best approach is the one that works best for you.

Programming Techniques for Speed

There are occasions when you can reduce the execution time of long running programs that are to be used many times. Under these conditions, different key sequences may result in faster and more efficient program operation.

When a program is running, the most time-consuming operations are program transfers. Therefore, minimizing the number of transfer statements leads to a faster running program. Although the use of subroutines is emphasized in earlier discussions, when program space allows, you may replace subroutines with in-line instructions to significantly increase speed.

Remember that a destination may be specified by an absolute address or by a program label. If an absolute address is used, the program pointer is immediately positioned at the new location. However, if a label is used, the calculator must search for the label until its location is found. Then, program execution is continued from that point.

When a program is initially entered into the calculator, it is difficult to know what the absolute addresses will be. Editing a program often causes addresses to change. The best procedure is to first write the program using labels and convert to absolute addressing after the program is completely debugged. Inserting addresses and deleting labels cause the addresses to change. However, this problem may be overcome using **[2nd] [Nop]** to reserve one program step with a label address that will be required for an absolute address later.

[2nd] [Nop] performs no operation when encountered in a program. Since this command does not interfere with execution (except when used as a label), it may be used as a space-holder. This technique is illustrated below.

Label Addressing Converted to Absolute Addressing

.	.
.	.
027 SBR	027 SBR
028 LNX	028 00
029 NOP	029 75
.	.
.	.
.	.
073 LBL	073 NOP
074 LNX	074 NOP
.	.
.	.
.	.
099 GTO	099 GTO
100 LNX	100 00
101 NOP	101 75

Note that location 075 is used as the absolute address since transferring to a label address positions the program pointer at the first location following the label.

The transfer instruction itself must also be reentered so as to instruct the calculator to automatically merge the address.

Use this key sequence for converting the previous example to absolute addressing.

[GTO] 29	[GTO] 74	[GTO] 101
[LRN]	[LRN]	[LRN]
[2nd] [Del]	[2nd] [Del]	[2nd] [Del]
[2nd] [Del]	[2nd] [Del]	[2nd] [Del]
[2nd] [Del]	[2nd] [Nop]	[2nd] [Del]
[SBR] 75	[2nd] [Nop]	[GTO] 75
[LRN]	[LRN]	[LRN]

Lesson 11—Sample Programs

This lesson contains several programs covering a variety of applications. You may find these programs to be a valuable source of problem-solving and programming techniques.

Compound Interest Program

If 5% interest per year is received on an account worth $1000, at the end of one year $50 in interest is added making the account worth $1050. The $1000 in the account today is called the "present value" of the account because it has received no interest. But at the end of one year you would expect it to be worth $1050 which is its "future value." Compounding interest means that once money is placed in an account and is left alone for more than one period, at the end of each period interest is added to what was in the account at the beginning of that period. Interest is also earned on interest such that the original $1000 is worth:

$1000 + $1000 (.05) = $1050 at the end of the first year
$1050 + $1050 (.05) = $1102.50 at the end of the second year

The percent interest rate is divided by 100 to obtain the decimal interest. Savings institutions use various periods in compounding interest (quarterly, daily, etc.). Flexibility may be added to the program by providing a means to tell the calculator how the compounding is done. By incorporating the number of compounding periods per year, the future value equation may be rewritten as:

$$FV = PV \times (1 + (i/100 \div c))^{cn}$$

The variables used above are:

FV = future value of investment
PV = present value of investment
i = annual interest rate (APR)
c = number of compounding periods per year
n = number of years of investment

In this case, enter the variables into user data memory. This allows the variables to be entered individually and makes it easier to evaluate different possibilities. Note that when a program is to be rerun using previously entered data, care must be taken to preserve the original data.

Investment Calculation Program

Press	Comments
[LBL] [A] [STO] [1] [R/S]	Define Label A as PV
[LBL] [B] [STO] [2] [R/S]	Define Label B as i
[LBL] [C] [STO] [3] [R/S]	Define Label C as c
[LBL] [D] [STO] [4] [R/S]	Define Label D as n
[LBL] [E]	Define Label E To Start Program
[RCL] [2] [÷] [1] [0] [0]	Convert i to Decimal Format
[÷] [RCL] [3]	Find Interest Per Compounding Period
[+] [1] [=] [y^x] [( )] [RCL] [3] [×]	Determine Compound Interest Factor for c × n Periods
[RCL] [4] [ )] [×] [RCL] [1] [=]	Multiply By PV To Find FV
[2nd] [Fix] [2]	Display FV Rounded To Cents
[R/S]	

Investment Calculation Program Listing

000 LBL	025 1
001 A	026 0
002 STO	027 0
003 01	028 /
004 R/S	029 RCL
005 LBL	030 03
006 B	031 +
007 STO	032 1
008 02	033 =
009 R/S	034 Y/X
010 LBL	035 (
011 C	036 RCL
012 STO	037 03
013 03	038 ×
014 R/S	039 RCL
015 LBL	040 04
016 D	041)
017 STO	042 ×
018 04	043 RCL
019 R/S	044 01
020 LBL	045 =
021 E	046 FIX
022 RCL	047 02
023 02	048 R/S
024 /	

User Instructions

Step	Procedure	Enter	Press	Display
1	Clear Program Memory		[2nd] [CP]	
1a	Partition Memory		[2nd] [Part] 55	71.54
2	Enter Learn Mode		[LRN]	ST
3	Enter Investment Calculation Program			
4	Exit Learn Mode		[LRN]	0

Variables May Be Entered In Any Order. There Is No Need to Reenter Variables That Do Not Change For New Problems

5	Enter present Value	PV	[A]	PV
6	Enter Annual Interest	i	[B]	i
7	Enter Number of Compounding Periods Per year	c	[C]	c
8	Enter Number of Years	n	[D]	n
9	Compute Future Value		[E]	FV

Find the future value of a $3,000 investment 5 years from now if the annual return rate is 8% compounded daily and compounded monthly.

Press	Display	Comments
3000 [A]	3000	PV
8 [B]	8	i
365 [C]	365	c
5 [D]	5	n
[E]	4475.28	FV
12 [C]	12.00	c
[E]	4469.54	FV

Pricing Control Program

Thus far we have used the calculator's user data memories primarily for storing and recalling variables. However, the calculator can add to, subtract from, multiply and divide the variables stored in user data memories without recalling them. Using the memory in this fashion is often referred to as memory arithmetic.

Assume a purchase order received in a business is comprised of different items in various quantities. In order to invoice the customer, multiply the quantity for each line item by its unit price to find the line item price. Then sum each line item price to determine the total order price. Additionally, to keep a record of the average unit price of each order, you total the line item quantities and divide the sum into the total order price.

Line Item	Quantity	Unit Price	Line Item Price
1	100	$0.25	$ 25.00
2	200	0.15	30.00
3	50	0.35	17.50
4	150	0.40	60.00
5	300	0.10	30.00
Total Order	800		$162.50
Order Avg. Unit Price		$0.203125	

To save time lost by displaying intermediate data, the data are stored in memories for recalling later, if desired. The cumulative order quantity is stored in memory 1, the cumulative order price is stored in memory 2, and the current average unit price is stored in memory 3. The program is designed to display only one intermediate result, the line item price of each line. The line item price is displayed after the quantity of an item and its unit price is entered. However, you may recall any of the other results whenever you need to see them.

One last note is that since the initial operations on memories 1 and 2 are to be sum instructions, the program should be equipped with an initialization routine which zeros these user data memories.

The solution can be described as follows.

1. Initialize memories 1 and 2.
2. Accumulate the total order quantity in memory 1.
3. Store the quantity of the current item in memory 4.
4. Multiply the unit price into memory 4 to determine the line item price.
5. Accumulate total order price in memory 2.
6. Memory 2 divided by memory 1 gives the average item price and is then stored in memory 3.

Pricing Control Program Listing

```
000 LBL     024 01
001 E       025 =
002 CMS     026 STO
003 CLR     027 03
004 FIX     028 RCL
005 02      029 04
006 R/S     030 R/S
007 LBL     031 LBL
008 A       032 B
009 SUM     033 RCL
010 01      034 01
011 STO     035 R/S
012 04      036 LBL
013 R/S     037 C
014 PRD     038 RCL
015 04      039 02
016 RCL     040 R/S
017 04      041 LBL
018 SUM     042 D
019 02      043 RCL
020 RCL     044 03
021 02      045 R/S
022 /
023 RCL
```

User Instructions

Step	Procedure	Enter	Press	Display
1	Clear Program Memory		[2nd] [CP]	
2	Enter Learn Mode		[LRN]	ST
3	Enter Pricing Control Program			
4	Exit Learn Mode		[LRN]	
5	Initialize Program		[E]	0.00
6	Enter Line Item Quantity	Quantity	[A]	Quantity
7	Enter Unit Price	Unit Price	[R/S]	Line Item Price
	Repeat Steps 6 and 7 for Each Line Item. After Each Line Item Entry the Following Variables May be Displayed:			
	Cumulative Quantity		[B]	Total Order Quantity
	Cumulative Cost		[C]	Total Order Price
	Average Unit Price		[D]	Average Unit Price

Now let's run the program using the data given earlier.

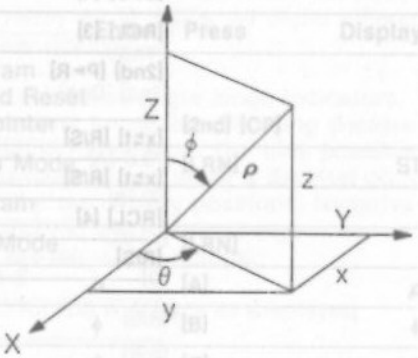
Press	Display	Comments
[E]	0.00	Initialize
100 [A]	100.00	Enter Quantity A
.25 [R/S]	25.00	Enter Unit Price A
		Show Line Item Price
200 [A]	200.00	Enter Quantity B
.15 [R/S]	30.00	Enter Unit Price B
		Show Line Item Price

50 [A]	50.00	Enter Quantity C
.35 [R/S]	17.50	Enter Unit Price C
		Show Line Item Price
150 [A]	150.00	Enter Quantity D
.4 [R/S]	60.00	Enter Unit Price D
		Show Line Item Price
300 [A]	300.00	Enter Quantity E
.1 [R/S]	30.00	Enter Unit Price E
		Show line Item Price
[B]	800.00	Total Order Quantity
[C]	162.50	Total Order Price
[D]	0.20	Avg. Unit Price (Rounded)
[INV] [2nd] [Fix]	0.203125	Avg. Unit Price (Exact)

Spherical Coordinates Program

Write a program to convert from spherical to rectangular coordinates.

$x = \rho \sin \phi \cos \theta$
$y = \rho \sin \phi \sin \theta$
$z = \rho \cos \phi$



Store ρ, ϕ, and θ in memories 1, 2, and 3 respectively. Place ρ in the T-register and ϕ in the display register. Find z by using [2nd] [P►R]. This conversion places $\rho \cos \phi$ in the T-register. Use this conversion again to find x and y after recalling θ to the display register. The program is designed to display x, y, and z by using the [R/S] key.

Press	Display	Comments
[LBL] [A]		Define Label A as ρ
[STO] [1] [R/S]		
[LBL] [B]		Define Label B as ϕ
[STO] [2] [R/S]		
[LBL] [C]		Define Label C as θ
[STO] [3] [R/S]		
[LBL] [D]		Define Label D To Calculate Coordinates.
[RCL] [1]		Convert (ρ ϕ) To (z, ρ sin ϕ)
[x≷t]		
[RCL] [2]		
[2nd] [P▸R]		
[x≷t]		Place ρ sin ϕ In t register and Store z In memory 4
[STO] [4]		
[RCL] [3]		
[2nd] [P▸R]		Convert (ρ sin ϕ, θ) To (x,y)
[x≷t] [R/S]		Display x
[x≷t] [R/S]		Press [R/S] To Display y
[RCL] [4]		Press [R/S] To Display z
[R/S]		

Spherical Coordinates Program Listing

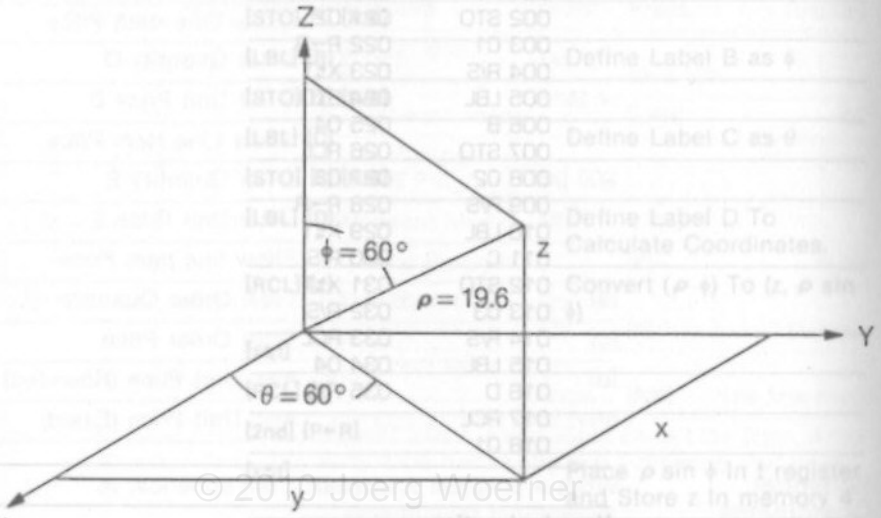
```
000 LBL    019 X⇆T
001 A      020 RCL
002 STO    021 02
003 01     022 P→R
004 R/S    023 X⇆T
005 LBL    024 STO
006 B      025 04
007 STO    026 RCL
008 02     027 03
009 R/S    028 P→R
010 LBL    029 X⇆T
011 C      030 R/S
012 STO    031 X⇆T
013 03     032 R/S
014 R/S    033 RCL
015 LBL    034 04
016 D      035 R/S
017 RCL
018 01
```


User Instructions

Step	Procedure	Enter	Press	Display
1	Clear Program Memory and Reset Program Pointer		[2nd] [CP]	
2	Enter Learn Mode		[LRN]	ST
3	Enter Program			
4	Exit Learn Mode		[LRN]	
5	Enter ρ	ρ	[A]	ρ
6	Enter ϕ	ϕ	[B]	ϕ
7	Enter θ	θ	[C]	θ
8A	Compute Coordinates and Display x		[D]	x
8B	Display y		[R/S]	y
8C	Display z		[R/S]	z

After entering the program, it may be tested with the following example.

Example: Convert $\rho = 19.6$, $\phi = 60°$, $\theta = 60°$ to rectangular coordinates.



Press	Display	Comments
[2nd] [Deg]		Place calculator in degree mode.
19.6 [A]	19.6	ρ
60 [B]	60	ϕ
60 [C]	60	θ
[D]	8.487048957	x
[R/S]	14.7	y
[R/S]	9.8	z

TEXAS INSTRUMENTS
INCORPORATED
Dallas, Texas

Printed in Japan

1052579-2