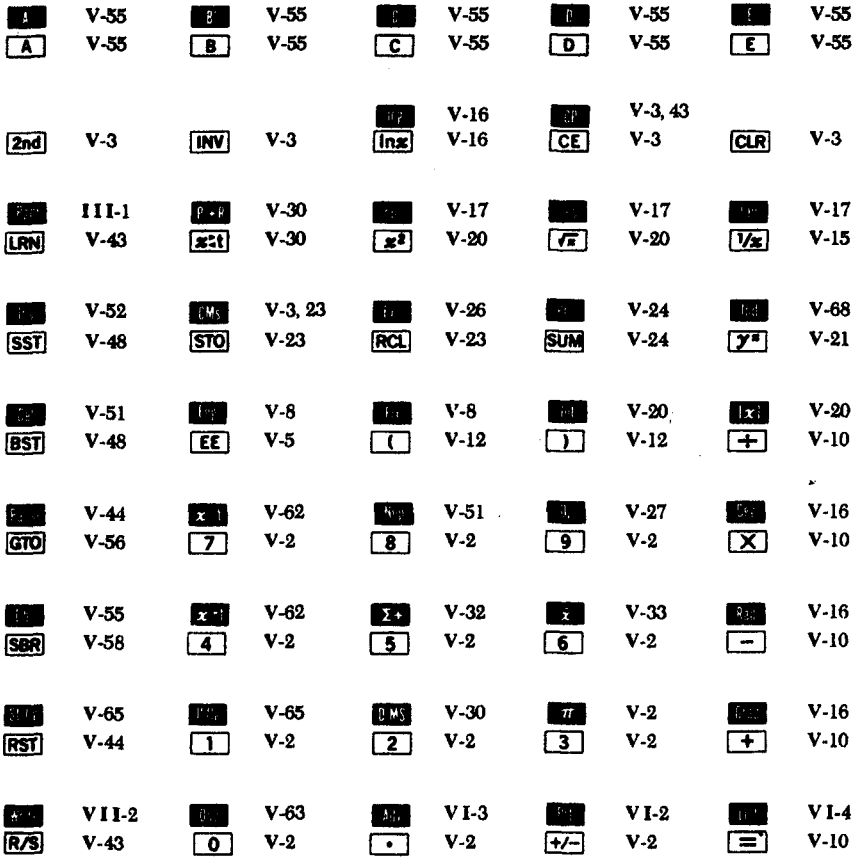
# TI PROGRAMMABLE 58/59

# Personal Programming

(EN)

# KEY INDEX

**This indexed keyboard provides a quick page reference to the description of each key.**




**IMPORTANT**

Record the serial number from the bottom of the unit and 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 PROGRAMMABLE –_____**

**Model No.** **Serial No.** **Purchase Date**

# TABLE OF CONTENTS

# TABLE OF CONTENTS (continued)

# TABLE OF CONTENTS (continued)

Section Page

# TABLE OF CONTENTS (continued)

Section Page

# TABLE OF CONTENTS (continued)

# I GETTING ACQUAINTED

## A QUICK LOOK

## INTRODUCTION

Today's handheld calculators can make it easy to utilize math in handling the problem situations that arise in many professional fields. A new speed, accuracy and confidence can now be part of our everyday life situations involving numbers and math from the routine to the most complex.

Initially, "the basics" — addition, subtraction, multiplication and division — were the only calculator capabilities available — and at the time, they were a revolutionary development. Next evolved calculators with more powerful math functions — squares, square roots, logarithms, trig functions, etc. These not only replaced the need for volumes of tables and charts, but also greatly increased the speed and accuracy possible in handling and solving the problems that arise in technical disciplines.

Now — a new dimension! Programmability in a handheld calculator opens the gateways to vast new problem solving areas — areas that only a computer could enter before. This manual 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.

This manual is written for both the TI Programmable 58 and 59. These two calculators differ only by the following.

| **TI Programmable 58** | | **TI Programmable 59** |
|---|---|---|
| | Registers available for | |
| up to 60 | Data Storage | up to 100 |
| up to 480 | Program Storage | up to 960 |
| | Reads and writes | |
| no | magnetic cards | yes |

All other calculator functions and operations are identical. Where the operations of the calculators differ, special notes have been made in the text. This book is organized like this:

- We'll start right in with some quick illustrations of just how easy using and programming your machine can be.
- After that we'll follow with a tour of the key's features and functions of your machine.
- Then we'll go into a step-by-step discussion of programming.
- Later in the manual we'll cover some of the more advanced programming features of your calculator, with a variety of application examples from various fields.
- The final section of the manual is a detailed and complete analysis of all calculator keys showing the full operating limits of the machine in various calculating situations. (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 that section and review your machine in technical detail.)

# Getting Acquainted

I

## A QUICK LOOK

### POWER UP

The battery pack furnished with your calculator was charged at the factory before it was sent out to you. However, due to self-discharging that happens in all batteries, the battery pack may require some charging before initial operation. If while you're first using your machine, the display becomes dim or erratic, the battery pack needs to be charged. Just turn the machine off, plug in your charger and wait a few minutes. Then proceed. You can be using your calculator while the battery pack is being charged.

Slide the ON/OFF switch to the ON position and you should see a single zero in the display. This shows that the battery is charged and the calculator is ready for action. Turning the calculator ON automatically clears your machine completely. To check your calculator's display, press the decimal point [ • ] and the change sign [ +/− ] keys, then press eight repeatedly to fill the display. An eight lights all segments of each digit position in the display. Note that the decimal point and minus sign progress to the left each time an eight is pressed. You can enter up to 10 digits into your calculator at any one time for either positive or negative numbers. All digit entries made after the tenth are simply ignored. Notice that the minus sign always stays immediately to the left of any negative number in the display for easy reading.

Whenever you exceed the limits of the display or ask the calculator to do something it cannot do, the calculator lets you know by flashing the display. This flashing is stopped by pressing the clear entry key [ CE ].

We'll be taking you on a "guided tour" of your machine, but remember that there's no substitute for just sitting down and exploring it on your own. This is one of the best ways to get to know what a versatile and powerful device it is. The more you learn about its far reaching capabilities, the better it is able to serve your needs.

### TYPES OF OPERATIONS

Basically, there are 3 types of operations your calculator can handle for you:

You can easily use one of the many *Solid State Software*™ programs built right into your machine to handle complex problems with a few keystrokes — OR —

You can teach your calculator your own problem-solving methods, and it can remember and execute them for you whenever you want — OR —

Your machine always stands ready to work as a high-powered manual calculator — ready to immediately handle "around-the-house" math as well as more intricate calculations with its advanced professional features.

## Running Library Programs

Without knowing anything about how to personally program your calculator, you can run many useful programs. A master library of prewritten *Solid State Software* programs is contained in a small module already inserted in the back of your calculator. This interchangeable module (other modules are available) contains a variety of general purpose programs described in the *Master Library Manual.* Through use of the program key Pgm , each program can be "called-up" and used according to the writeup in the manual. To illustrate how easy these programs are to use, let's play the "Hi-Lo" game.

The object of the game is for you to guess a secret number in as few guesses as possible. The calculator chooses a number in the range from 1 to 1023. The calculator responds to each of your guesses with a "too low," "too high," or "correct". Your score (number of guesses) is tallied by the calculator. Follow the User Instructions and play.

**USER INSTRUCTIONS**

| Step | Procedure | Enter | Press | Display |
|---|---|---|---|---|
| 1 | Select program | | 2nd Pgm 21 | |
| 2 | Key in a decimal point [ • ] and a series of random digits | Your Number | A | Your Number |
| 3 | Generate secret number | | B | 0. |
| 4 | Enter your guess (1 to 1023)<br>Clue: -1. guess is low<br>1. guess is high<br>flashing 0. guess is correct | Guess | C | Clue |
| 5 | Repeat Step 4 as needed | | | |
| 6 | Display score (number of guesses) | | D | Score |

Most prerecorded programs are as easy to use as this. Actually dozens of program steps have been executed, but these are automatically handled for you. All you have to do is enter the numbers you want to work on, and start the program.

Here's an important point — the key to using any of the *Solid State Software* programs is the library manual. All of the ins, outs and details you need to get the most out of each library program are included in the manual. So, refer to it whenever you're using any prerecorded program. It might be a good idea to glance through your *Master Library Manual* right now. Get a feel for the programs that are already at your disposal — ready to help handle problems for you.

# Getting Acquainted

I

## A QUICK LOOK

### Calculations From the Keyboard

Your advanced TI calculator is equipped with *AOS*™ method of entering problems, one of the most straightforward entry methods yet devised. Problems are easily solved by entering them directly into the calculator in the order they are written, left to right. For instance, to convert 100°C, 36°C, and −4°C to Fahrenheit, you need to multiply the Celsius reading by 9/5 and add 32. °F = °C X 9/5 + 32.

| Press | Display |
|---|---|
| 100 [X] | 100. |
| 9 [÷] | 900. |
| 5 [+] | 180. |
| 32 [=] | 212. |

You can repeat this sequence to find 36°C = 96.8°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 a calculation sequence has been determined and you have several values that need the same 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:

[X]
[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 is ready to perform this series of operations on any number (here, any Celsius reading) that you may enter into the display.

## A QUICK LOOK

You're now ready to have your machine do the Celsius to Fahrenheit calculations on any number you enter.

- Key in your Celsius value
- Press [RST] (reset) to tell the calculator to start at the beginning
- Press [R/S] (run/stop) to begin program execution

| Press | Display |
|---|---|
| **100** [RST] [R/S] | 212. |
| **36** [RST] [R/S] | 96.8 |
| **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 you've just seen in action — the 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

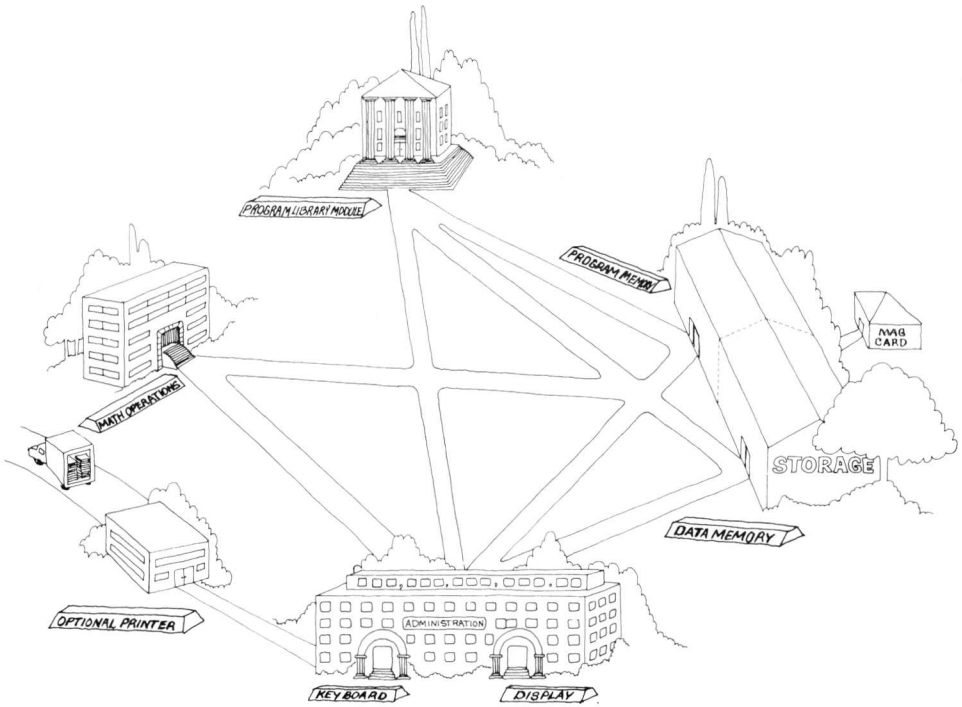
Your calculator is compatible with an optional printing unit, the PC-100A. The printer can record the displayed value on paper whenever you tell it to. When solving problems directly from the keyboard, you can selectively print any or all desired intermediate results or provide a complete listing of a stored program. When executing a program, print instructions encountered in the program cause automatic printing of the quantity in the display register. These printing features allow you to run a program while recording multiple answers. The trace option on the printer prints all steps performed and the corresponding numerical results.

Through use of the special control operations you can assemble and print any messages you need to identify segments of the listing or for titles. Up to 20 characters can be printed per line, made up from a master set of 64 characters.

## A QUICK LOOK

# CALCULATOR CAMPUS

Your new calculator has many powerful structures that work together to form a "community of benefits" all designed to work as an easy to use problem-solving system. In this section you've had a very quick look at some of these features in action. In the next section we'll take a brief "guided tour" of this working community in a little more detail, and then get right on into the business (and pleasure) of programming.



# II A "GUIDE KEY TOUR"

## A LOOK AT THE FEATURES AND FUNCTIONS

Before you plunge into some of the more advanced features of your machine, it will be useful to come along on a brief tour of the main features and functions available on the keyboard. This is particularly true if this is your first experience with an advanced calculator. Many calculator owners never fully access all the power available in their machines — simply because they've never taken the time to see each key in action. In this section you'll get a quick key review — requiring only about 10-15 minutes of your time. This will generally familiarize you with the main keyboard features — so that as you move on in to programming, you'll be able to take full advantage of all that the machine can do.

A note to various users:

If you're already familiar with advanced calculators with *AOS* entry method, you may want to skip this key tour section and get right into programming (Section IV).

For a *specific and detailed* description of all the calculator's various operations and capabilities, refer to Section V for an in-depth discussion of each key and feature.

As you proceed through this tour, be sure your calculator is out and handy. Check out each key and feature as it's discussed. The best way to learn about your machine is to use it!

# A "Guided Key Tour"

II

## A LOOK AT THE FEATURES AND FUNCTIONS

## KEYBOARD BASICS

### Clearing the Display — [CE], [CLR]

There are two procedures that allow you to clear the display register of your calculator depending upon your needs as you proceed through a problem.

[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 may also be used to stop a flashing display created by an error condition.

[CLR] GENERAL 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 removed.

### Data Entry Keys — [0] – [9], [•], [+/–], [π]

Numbers are entered into the machine with the data entry keys [0] – [9] [•] [+/–]. As you enter any number, the decimal point stays 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 push the change sign key [+/–] once. (Pressing [+/–] again changes the sign back again).

Pressing [2nd] [π] places the first 10 digits of $\pi$ in the display 3.141592654. 13 digits are carried in the internal display register, 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 *AOS* entry method which makes problem solution with these keys exceptionally easy. Basically, you just key in the problem the way it's written, press [=] and get your result. The amazing feature of the *AOS* entry method is that it automatically sorts out mixed operations in a problem for you, and applies them in the correct order as it calculates your result. (We'll say more about the *AOS* entry method on the next page.)

When you press the [=] key, all pending operations (things waiting to happen inside your calculator), are completed. You get your result, and the calculator is cleared — ready to start on the next problem.

Example: Calculate $15 + 7 \times 31 - 4 = ?$

**Press: 15 [+] 7 [×] 31 [–] 4 [=]** **Display: 228**

NOTE: Observe that *AOS* entry method instructed the calculator to interpret the expression as $15 + (7 \times 31) - 4$, where $7 \times 31$ is calculated, then added to 15 and 4 subtracted from this.

## The *AOS* Entry Method

Mathematics is a science which adheres to a clearly defined set of rules. One such rule is that it never permits two different answers to the same series of operations. Because of this requirement — one solution for any computation — mathematicians have established a universally accepted set of rules when mixed operations are used in one calculation. For example: the problem:

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

has only one right answer! (Know what it is? It's 9.)

You can key this problem directly, left to right into your calculator and you'll get the correct answer. The algebraic hierachy 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 it received from you in the following universally accepted order:

1) Special Single Variable function keys — act on the displayed number immediately — as soon as you push the key. (We'll talk more about each of these keys later in the "tour" — but they include all the keys for the trig and log functions and their inverses, as well as square and square root, reciprocal, and conversions.)

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

3) Multiplications and divisions are completed, followed by

4) Additions and subtractions.

This algebraic hierarchy applies to each set of parentheses.

Finally, the equals key completes all operations.

There are cases in problem solving where *you* want to be the one who specifies the order in which an expression is evaluated. In these cases you can control the order with the parentheses keys, [ ( ] [ ) ], which are discussed in the next section. Parentheses demand a special first level of attention in mathematics — and they're treated that way by your calculator.

# A "Guided Key Tour"

II

## A LOOK AT THE FEATURES AND FUNCTIONS

### Parentheses Keys — [(], [)]

In a variety of problems, you may need to specify the exact order in which expressions are evaluated, or the way in which numbers are grouped, as a problem is solved. Parentheses give you a way to cluster numbers and operations. By putting a series of numbers and operations in parentheses you tell the calculator "Evaluate this little problem first — down to a single number result, then use this result for the rest 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 in your mind about how the calculator will handle an expression. Your calculator can have as many as 9 parentheses sets open at any one time with as many as 8 operations pending. The following is an example of this full capability.

$$(((2\times(2\times(2\times(2\times(2\times(2+2y^{x}(2+.2))-(2+2)))))) \div 2)\div 2)$$

As you key in this sequence, note that no calculations take place until the first closed parenthesis is keyed in. Your calculator remembers all instructions keyed in and interprets them when it's supposed to.

Note: an important point when using parentheses. You may often see equations or expressions written with parentheses to imply multiplication: (2 + 1) (3 + 2) = 15. *Your calculator will not perform implied multiplications.* You have to key in the operation between the parentheses:

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

Here's an example using parentheses:

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

In problems of this type — you want the calculator to evaluate the entire numerator, then divide by the entire denominator. You can be sure of this taking place by placing 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. |

## Dual 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 right 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 right below the function.

For example, to find the natural logarithm of a number, press [ln x]. To find the common logarithm of a number, press [2nd] [ln x]. In order to distinguish the second function key, in this manual shows it as [2nd] [log]. First function operations, therefore, are indicated by [ ]. Second functions are indicated by [2nd] [ ].

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, the purpose of that function or key is reversed. The [INV] key works together with quite a few keys on your calculator to provide extra functions, or to reverse an operation.

The [2nd] and [INV] keys allow 108 different keyboard operations to be performed using only 45 keys. For use with specific keys, see *Dual Function Keys* in Section V.

# A "Guided Key Tour"

II

## A LOOK AT THE FEATURES AND FUNCTIONS

### Memory Keys — CMs , STO , RCL , Exc

Each time you turn on your calculator there are 60 data registers available (30 for the TI Programmable 58) for you to use. Actually, the number of data registers available versus the amount of program memory is variable. (See *Selection of Memory Size* in MEMORY CAPABILITIES in Section V for complete details.) Data registers are special locations in the calculator where you can store numbers you may need to use later.

Because there is usually more than one data register available for your use, you must indicate which register you want to use by specifying its two-digit number **XX**. For example, STO **08.**

The CE and CLR keys do not affect what is in the memories; however, pressing 2nd CMs clears all data registers simultaneously (places a 0 in all registers).

STO **XX** — STORE — This instruction stores the number held in the display register into data register **XX**(00-99) without disturbing the contents of the display register. (Any number previously stored in register **XX** is cleared out first.)

RCL **XX** — RECALL — This instruction simply brings the contents of data register **XX** to the display register. Again, the contents of data register **XX** are not disturbed.

2nd Exc **XX** — MEMORY EXCHANGE — The exchange sequence simply swaps whatever is in data register XX with the contents of the display register. (The display register value is stored in register **XX** while the number stored in memory is called to the display register.) This key is handy in many situations allowing you to make a quick check or use what is in memory without losing what's in the display register.

Example: Store and recall 3.21.

| Press | Display | Comments |
|---|---|---|
| **3.21** STO **08** | 3.21 | Store 3.21 in register 8 |
| CLR | 0 | Clear display |
| RCL **08** | 3.21 | Recall contents of register 8 |

Example: Evaluate: (A + 2) + A(A + 2) for A = 9.3069128.

| Press | Display | Comments |
|---|---|---|
| [CLR] | 0 | Clear any calculations in progress. |
| **9.3069128** [STO] **12** | 9.3069128 | Stores A in register 12. |
| [+] **2** [+] | 11.3069128 | A + 2 is evaluated. |
| [2nd] [Exc] **12** | 9.3069128 | Stores A + 2 in register 12 and calls A to the display register. |
| [X] [RCL] **12** | 11.3069128 | Recalls A + 2 to the display register. (Note that a [X] must be between A and A + 2) |
| [=] | 116.5393643 | Completes all pending operations to arrive at the final result. |

Note that the long value of A only had to be entered once, saving time and possible errors. The exchange key performs the task of a store and a recall also saving calculation effort.

## Memory Arithmetic Keys — [SUM], [Prd]

There is also a series of key sequences that let you operate on the numbers stored in memory without affecting other calculations in progress:

[SUM] **XX** — MEMORY SUM — This sequence allows you to add whatever is in the display register directly to what's stored in register **XX**. The result of the addition is stored in the memory while the display register is unaffected. Similarly, the sequence [INV] [SUM] **XX** subtracts the value in the display register from the contents of register **XX**.

[2nd] [Prd] **XX** — MEMORY PRODUCT — This sequence causes the contents of register **XX** to be multiplied by the display register value while [INV] [2nd] [Prd] **XX** divides register **XX** by the number in the display register. Again the result is left in memory and the display register is undisturbed.

These instructions perform similar to the way basic arithmetic operations do in normal keyboard calculations, except that results are accumulated in a data register instead of the display register.

Example: Find the total cost of items of $28 and $6.60 with 5% sales tax.

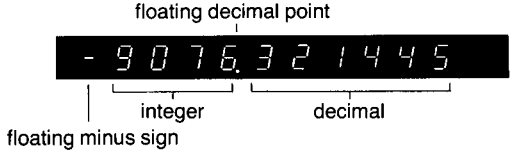
| Press | Display | Comments |
|---|---|---|
| **28** [STO] **01** | 28. | Store 28 in data register 1 |
| **6.6** [SUM] **01** | 6.6. | Add 6.6 to data register 1 |
| **1.05** [2nd] [Prd] **01** | 1.05 | Multiply data register 1 by 1.05 |
| [RCL] **01** | 36.33 | Total Cost |

## A LOOK AT THE FEATURES AND FUNCTIONS

# DISPLAY CONTROL

## Standard Display

The display provides numerical information complete with negative sign and decimal point and flashes on and off for an overflow, underflow, or error condition. (A complete list of error conditions is found in Appendix B.) An entry can contain as many as 10 digits. All digits entered after the tenth are ignored.




**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 results 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. × $10^{11}$.

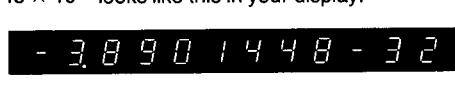

## Scientific Notation Key — EE

In many applications, particularly in science and engineering, you may find yourself needing to calculate with very large or small numbers. Such numbers are easily handled (by both you and your calculator) using scientific notation. A number in scientific notation is expressed as a base number (mantissa) times ten raised to some power (exponent).

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

To enter a number in scientific notation
Enter the mantissa using up to 8 digits — (then press +/− if it's negative).
Press EE (Enter Exponent) — "00" appears at the right of the display.
Enter the power of 10 (then press +/− if it's negative).

A number such as −3.8901448 × $10^{-32}$ looks like this in your display:




In scientific notation the power of ten tells you where the decimal point would have to be if you were writing the number out in longhand.

A positive exponent tells you how many places the decimal point should be shifted to the right, a negative exponent — how many places to the left.

Example: $2.9979 \times 10^{11} = 299{,}790{,}000{,}000$
(Move decimal 11 places to the right and add zeros as needed)
$1.6021 \times 10^{-9} = 0.0000000016021$
(Move decimal 9 places to the left and add zeros as needed)

Once you initiate the scienfitic notation format it stays there until you deliberately 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 this format when it clears the display.

## 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, $10^{-6}$ 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.

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 digits you'd like to appear in the display to the right of the decimal point as you go through your calculations. Just press [2nd] [Fix], then press the desired number of decimal places (0 to 8). The calculator then rounds all subsequent results to this number of decimal places for display only. However, you may still make entires with as many digits as you like as the calculator retains its own internal (13 digit) accuracy. [INV] [2nd] [Fix] removes fix-decimal.

Example: $2 \div 3 = .6666666667$

| Press | Display |
|---|---|
| [CLR] | 0. |
| 2 [÷] 3 [=] | .6666666667 |
| [2nd] [Fix] 6 | 0.666667 |
| [2nd] [Fix] 2 | 0.67 |
| [2nd] [Fix] 0 | 1. |
| [INV] [2nd] [Fix] | .6666666667 |

# A "Guided Key Tour"

II

## A LOOK AT THE FEATURES AND FUNCTIONS

## ALGEBRAIC FUNCTIONS

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

These three easily accessible keys are essential for speedy handling of a variety of equation solving situations. All three of these keys act immediately on the number held by the display register without affecting other calculations in progress.

[x²] — SQUARE — Calculates the square of the number, x, in the display register.

[√x] — SQUARE ROOT — Calculates the square root of the number, x, in the display register.

[1/x] — RECIPROCAL — Divides 1 by the display register value x.

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]

This powerful key allows you to raise any positive number to a power. You may also use this key to find the roots of a positive number.

**For Powers ($y^x$)**

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

Example: Calculate $2^6$.

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

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

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

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

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

NOTE: You should only enter positive values for y, a flashing display results for negative entries.

## Logarithms — [ln x] , [log]

Logarithms are mathematical functions that enter into a variety of technical and theoretical calculations. Basically, if $x = y^2$, then ln x (to the base y) = 2. The keys discussed below give you immediate access to the logarithms of any positive number — without affecting calculations in progress — and without having to deal with bulky tables.

[ln x] — NATURAL LOGARITHM — Immediately calculates the natural logarithm (base e = 2.718281828459) of the number held in the display register. (A flashing display results if this number is negative or zero.) The antilogarithm of the natural log ($e^x$) is found using the sequence [INV] [ln x] .

[2nd] [log] — COMMON LOGARITHM — Immediately 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] .

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] [ln x] | 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. |
| [ln x] | 3.425195888 | The result. |

## A LOOK AT THE FEATURES AND FUNCTIONS

### Angular Mode Keys — Deg , Rad , Grad

Your calculator is equipped to handle a variety of calculations that involve angles — notably the trigonometric functions and polar/rectangular conversions. Angles can be measured 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:

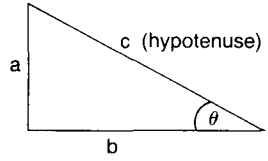
2nd Deg — SELECT DEGREE MODE — In this mode all entered and calculated angles are measured in degrees, until another mode is selected. (One degree = 1/360 of a circle — a right angle equals 90°.)

2nd Rad — SELECT RADIAN MODE — In this mode all angles are measured in radians (one radian equals 1/2 $\pi$ of a circle — a right angle equals $\pi/2$ radians).

2nd Grad — SELECT GRAD MODE — In this mode all angles are measured in grads (one grad equals 1/400 of a circle — a right angle equals 100 grads).

### Trigonometric Keys — sin , cos , tan

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




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

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

The sequences INV sin , INV cos , and INV tan are used to calculate the inverses of these functions. The resulting angles are displayed in units corresponding to the selected angular mode.

In the degree mode, all anges are interpreted in decimal format. (See Degree Format Conversions in Section V.)

# CONVERSIONS

## Degree Format Conversions — D.MS

There are two ways of representing an angle in degrees.

One method is to use the decimal degree format DDD.dd. Here DDD represents the integer portion of the angle while .dd denotes the fraction portion written as a decimal. (You may use as many as 10 digits.)

The second method is to use the degree.minute-second format DDD.MMSSsss. Again, 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. Observe that the decimal point separates the degrees from the minutes.

To convert from the degree.minute-second format to decimal degrees enter the angle into the display (DDD.MMSSsss) and press [2nd] [D.MS]. Pressing [INV] [2nd] [D.MS] reverses the conversion process and converts decimal degrees to degrees, minutes and seconds.

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

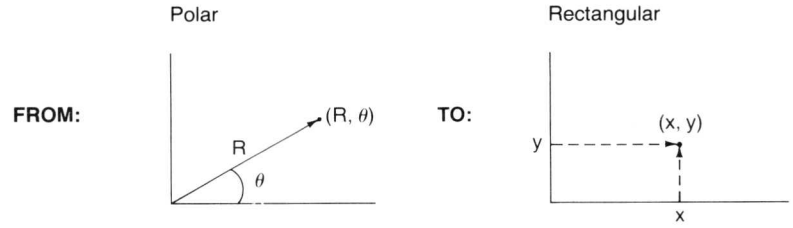
Example: Convert 54°02'09.6" to its decimal equivalent and back.

| Press | Display | Comments |
|---|---|---|
| **54.02096** [2nd] [D.MS] | 54.036 | DD.ddd |
| [INV] [2nd] [D.MS] | 54.02096 | DD.MMSSs |

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

## Polar/Rectangular Conversions — P→R

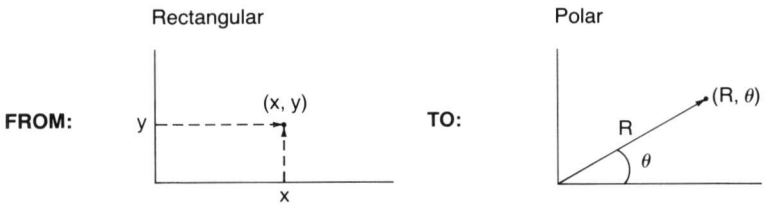
This is an especially handy feature of your calculator that is particularly useful in science and engineering applications. Working with the [x⇄t] key — it's fast and easy to convert from polar to rectangular coordinates, or vice versa. Just follow the key sequences illustrated below:




To convert from polar to rectangular coordinates:

- Enter your value for "R"
- Press [x⇄t]
- Enter your "θ" value (be sure angular mode is correct)
- Press [2nd] [P→R] to display "y"
- Press [x⇄t] to display "x"

## A LOOK AT THE FEATURES AND FUNCTIONS

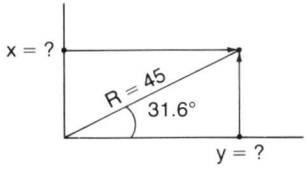



To convert from rectangular to polar coordinates:

- Enter your "x" value
- Press [x⇄t]
- Enter your "y" value
- Press [INV] [2nd] [P→R] to display "$\theta$" in selected angular units
- Press [x⇄t] to display "R"

"R" is now displayed.

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** | 31.6 | Place degrees in display |
| [2nd] [P→R] | 23.57936577 | Convert to rectangular coordinants 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 discussed in various programming sections.



## STATISTICAL FUNCTIONS ON KEYS

### Mean, Variance and Standard Deviation

You may find yourself handling large sets of data points describing some particular factor or parameter of a large number of items. (This 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 is — a measure of how far the data differs from the mean.

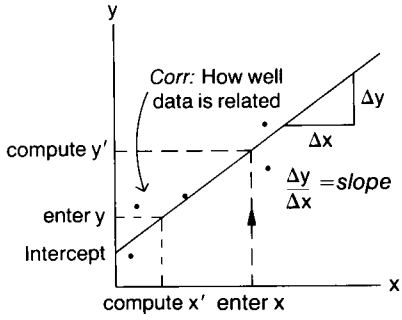
Refer to *Statistics* in Section V for a complete discussion of how to use these powerful functions.

## Linear Regression

Linear regression may sound like a highly technical or threatening title to you — but it's a process that your calculator makes very easy to use. And — it's one that deals with one of the oldest problems in the world — predicting future events.

In the linear regression situation, you usually have data expressed as pairs of variables that you could plot 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 "best straight line" through your data points. You may then use the straight line to make predictions.

Once the data is entered your calculator is ready to draw the best straight line through your points and give you the following information from it:




Statistics involving non-linear curve fits, exponential population plots for example, can also be accommodated by the calculator.

The use of these and other features is detailed in *Statistics* in Section V.

# III USING "BUILT-IN" PROBLEM SOLUTIONS

## ACCESSING *SOLID STATE SOFTWARE* PROGRAMS

The term "software" may or may not be familiar to you — and actually it has a variety of definitions. Basically, it refers to the instructions and programs — things that usually can be written on paper — that tell a computer or calculator what to do, and a user like yourself how to use them. Your calculator has a provision for an assortment of easy-to-use but extremely powerful programs to be inserted into the calculator and used by you at the touch of a key. These programs — especially written to handle user needs in a variety of fields — are stored in a special library module in back of your machine. This module can be easily replaced with a different module. The program information is all contained in a tiny solid-state "chip" of silicon — similar in construction to the silicon integrated circuit that is the heart and brain of a calculator. Hence the term *"Solid State Software"* programs. There's a lot of "software" — programs with easy-to-use features stored for you in your *Solid State Software* module. The advantages

- — a lot of program capability is packed in little space — easy to carry and use.
- — library programs are accessible from keyboard anytime.
- — the library programs are especially written to be easy to use, even by the beginning calculator user.

To complete the effectiveness of each module, there is a library manual. All information peculiar to each program is found in an easy-to-use format in this manual.

## PROGRAM LIBRARIES

A Master Library that is a basic assortment of useful programs comes with your calculator. Other professional libraries can be obtained from most TI retailers or ordered directly from Texas Instruments. Each library contains a selection of programs that make it easy to use some of the powerful mathematical techniques of the various professional fields. A library consists of a program library module, a manual explaining in detail the use of each program in that library, a storage case and a set of program label cards.

## PROGRAM LIBRARY MODULE

The programs of each library are stored in library modules, one module for each library. A module can be easily inserted into the back of the calculator and used immediately. Modules are durable devices, but should be handled with care for long life.

**CAUTION**

**Be sure your body is free of static electricity before handling any module.**

This is especially true when the charger is connected because this grounds the calculator. Just touch some metal object to electrically disharge your body. The contents of a module can be severely damaged by static discharges. See Appendix A for more on maintaining the modules.

# Using "Built-In" Problem Solutions

III

## ACCESSING *SOLID STATE SOFTWARE* PROGRAMS

The Master Library module is installed in the calculator at the factory, but can easily be removed or replaced with another. It is a good idea to leave the module in place in the calculator except when replacing it with another module. Be sure to follow these instructions when you need to remove or replace a module.

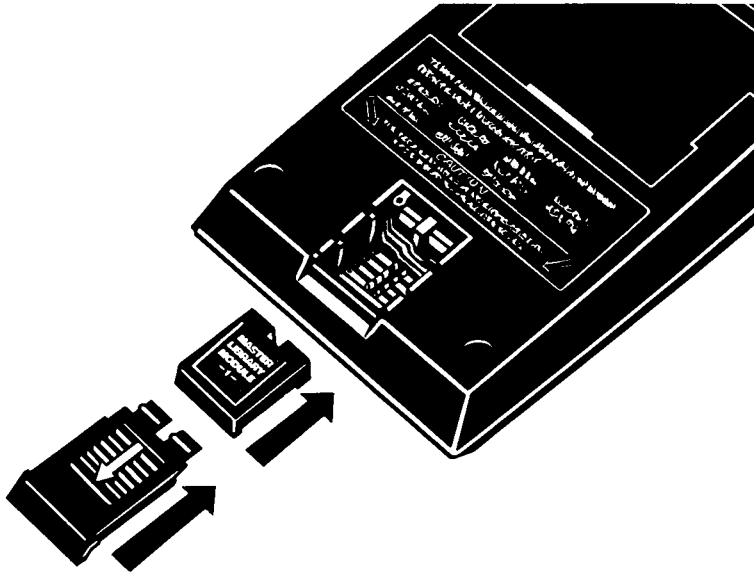
1) Turn the calculator OFF. Loading or unloading a module can short the contacts, seriously damaging the module and/or the calculator if the calculator is ON during loading or unloading.

2) Slide out the small panel covering the module compartment at the bottom of the back of the calculator. (See Diagram below) Again, eliminate all static charges before handling the module.

3) Remove the module. You may turn the calculator over and let the module fall out into your hand.

4) Insert the module, notched end first with the labeled side up into the compartment. The module should slip effortlessly into place.

5) Replace the cover panel, securing the module against the contacts.



**Avoid any action that could bend, contaminate or otherwise damage the contacts.**

## RUNNING LIBRARY PROGRAMS

Once a library module has been loaded into the calculator, calling a particular program is done simply by requesting it by its number (each program in a library has its own number). The required sequence is [2nd] [Pgm] **mm**, where **mm** is the two-digit number assigned to the program you have chosen to run.

To use a program, carefully follow the instructions in the library manual. A program label card is included for each program. This nonmagnetic card specifies the user-defined key assignments and can be fitted into the window above these keys once it is separated from the sheet of labels.

Example: What is the value of $1000 after 20 years of compounding at an 8% interest rate?

The Compound Interest program in the Master Library can readily solve this problem.

| Press | Display | Comments |
|---|---|---|
| [CLR] [2nd] [Pgm] **18** | 0. | Calculator goes to Compound Interest Program (PGM 18) |
| [2nd] [E'] | 0.00 | Initialize |
| **20** [A] | 20.00 | Enter number of periods |
| **8** [B] | 8.00 | Enter interest rate |
| **1000** [C] | 1000.00 | Enter present value |
| **0** [D] | 4660.96 | Calculate future value |

You can use this program over and over once you are there without having to use the PGM instruction again. Press [RST] or [2nd] [Pgm] **00** to return to program memory and keyboard operation or you can run another library program through use of the [Pgm] key.

Library programs can also be called by other programs as explained in *Subroutines* in Section IV. This feature greatly expands the programming capabilities of your calculator.

If you want to find out which library is in the calculator at any time without having to open the module compartment, press [2nd] [Pgm] 1 [SBR] [2nd] [Write] ([2nd] [Pgm] 1 [SBR] [2nd] [R/S] for the TI Programmable 58) and the module number is displayed (and printed along with the module name if you have the calculator on a printer).

## ANALYZING LIBRARY PROGRAMS (DOWNLOADING)

Normally, library programs are confined to their module for ready access, whenever needed. When a program is used, processing actually enters the module and performs its task. To gain access to the library program, you can bring it into program memory. Now all the calculator's programming tools can be used to analyze the individual steps and alter the program to your particular needs if necessary. Actually, only a copy of a program is brought into program memory, the module contents can never be changed. The procedure to "download" a program is easy.

1) Verify that there is sufficient program memory space available for the incoming program. See *Partitioning* in the next section.

2) Press 2nd Pgm **mm** to designate which program to download.

3) Press 2nd Op **09** to download the program.

This procedure places the requested program into program memory beginning at program location 000. The downloaded program writes over any instructions previously stored in that part of program memory. Therefore, a program in program memory cannot download a module program.

Once in program memory, the program can be manipulated for whatever purposes you need. You cannot, however, place the altered program back into the library module. If you need to preserve the "new" program, you can write down each step on a coding form, record it onto a magnetic card (or cards) if you have a TI Programmable 59 or list it if you have an optional printing unit.

A library module containing proprietary programs can be protected from downloading. A request to download one of these programs flashes the display.

If a library module does not perform as expected, see Appendix A.

# IV PROGRAMMING CONSIDERATIONS

## WHAT IS PROGRAMMING?

Computers are having such an impact on everyday life that we've become familiar with terms such as *computer-programmers, programming the computer, programming language,* or just plain *programming.* For some people, these terms conjure up visions of super-sophisticated individuals dealing in a highly complex field and just the thought of becoming a programmer is beyond the realm of possibility, at least without a great deal of training.

Not so, the era of personal programming is here. In fact, calculator programming is simple, and most intriguing is the fact that anyone can be programming calculators after a couple of easy lessons. Texas Instruments programmable calculators are designed to make programming simple and easy. Your calculator is versatile enough to allow you to enjoy the speed and power that programming offers — whether you are someone using just basic arithmetic, or an aerospace engineer working with extremely complex mathematics. The calculating power is there for everyone, but use only what is required for your application. You'll be amazed at how quickly and easily time-consuming problems can be solved with simple arithmetic and simple programs working together.

*Programming* is logical thinking. In simplest terms, a program is a set of instructions telling a machine or a person how to do something. A *calculator program,* therefore, tells a calculator how to do something, in particular how to perform calculations. When you want your calculator to do a job all you really need to do is to tell it exactly what you want it to do and how you want it done. A *program* is a list of precise instructions in specific order to be executed faithfully in a literal way.

A *language* is merely the means by which you can communicate with your calculator. There is even a language to communicate with the simplest four-function calculator. Applied to programming, a language is a necessary means to communicate your program to your calculator.

A *calculator language* is heavily weighted towards common sense and the use of arithmetic. If, therefore, you have experience in carrying out arithmetic calculations, either with pencil and paper or on a calculator, you already know most of your calculator's programming language. The functions explained in this manual for keyboard operation can be used in the same way in programs.

A calculator (like any computer) performs with literal faithfulness those and only those instructions given it. This characteristic makes working with these machines a mixed experience. The result is that you, the programmer, have to be careful what you tell the calculator to do and the order in which you tell it to complete the instructions. A calculator does exactly what you instruct it to do, regardless of whether you want it done that way or not. The techniques we discuss here will allow you to start realizing the potential of your calculator and make you a functional part of the era of personal programming.

# Programming Considerations

# IV

## ELEMENTARY PROGRAMMING

### Placing a Variable in a Program

Consider the following simple expression:

$$A + B = C$$

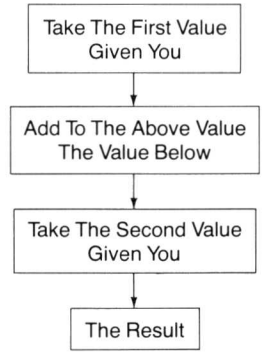
When using a basic four-function calculator, the values of A and B cannot be identified at a later time, they must be known at the time the expression is keyed in. After this first expression is keyed in and a result is obtained, to change either or both values you have to key in the entire expression again. With the programmable calculator you may key in the instructions leaving the values undefined and then get an answer at a later time by keying in only the values to be changed.

Now with the simple arithmetic expression above, the four-function calculator may be just as fast to use as a programmable calculator. The real advantage of the programmable calculator can be seen in an expression like the one below. Let's say you're in a situation where an answer is needed for ten different values of A, assuming B and C do not change.

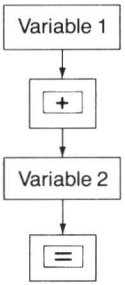
$$A \times (B \div (1 + A)^{-C})) = \text{RESULT}$$

You'd like to be able to enter the equation just once, then change only the value of A for each calculation. With your programmable calculator it's easy to do just that.

Let's go back and work with our simple expression again. Consider how to give instructions to the calculator. First, write the instructions as if to instruct another person, and then convert them to calculator instructions.




A and B are values that can be anything — they can vary. These values are often called variables.

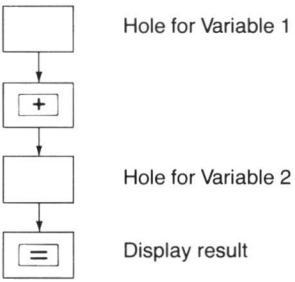



As far as the calculator is concerned, if you don't enter the variable as part of the program, two things must be done:

1. Leave a *hole* at the right spot in your instructions where you can place the variable at a later time.

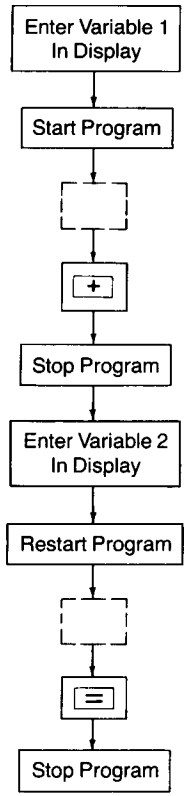
2. Tell the calculator where to look for the variable value when it needs it. You can instruct the calculator to look for a variable either in the display or in one of its data registers.

Here we'll redraw the instruction sequence, leaving *holes* where the variables should be inserted.




Now, if the calculator needs to use a displayed number as a variable, leave the hole empty. When the calculator starts running through a program, whatever value is in the display is placed in the first hole. A value for the second variable must also be found in the display, so we'll need to stop the program just before the variable is needed and enter it into the display. Then when the program is restarted the calculator takes the displayed value and continues.

One technique that lets you leave holes in your program for new entries or data is to simply stop the program at that point with the R/S (Run/Stop) key. You're just telling the machine to *hold everything* so that you can key the next value it needs into the display. (We can call this an *implied hole* since no gap is actually left between program instructions. By stopping execution at some point in the program we're implying that we want to do something at that point — make a data entry.)




From the Keyboard

From the Keyboard

Implied Hole-Takes Variable 1 From Display and Inserts Here

Program Instruction

Program Instruction — Stop So Next Variable Can Be Entered

From the Keyboard

From the Keyboard

Implied Hole — Takes Variable From Display and Inserts Here

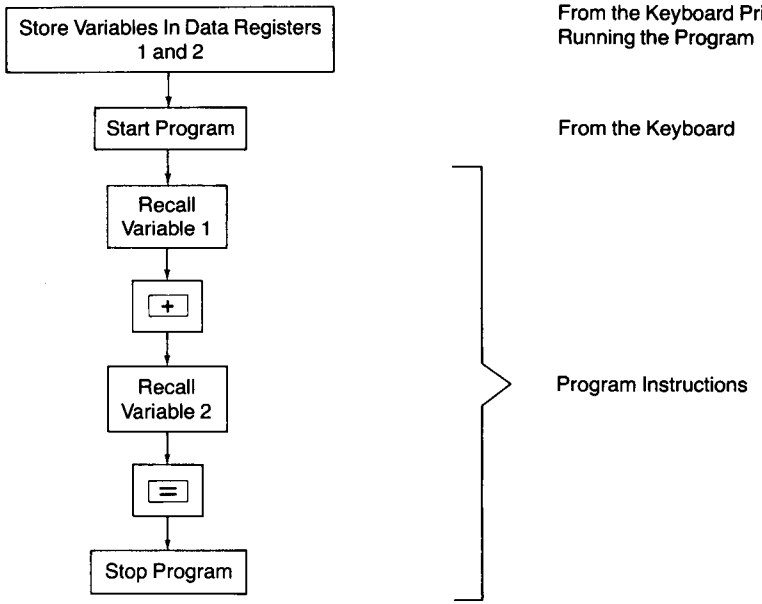
Program Instruction

Display Result

**Variables in the Display Flow Chart**

The above method (stopping the program to enter data) is ideal when a completely new set of variables is to be used each time the program is run. You may find another technique preferable when only one value needs to be changed. In this procedure you use the calculator's data registers to store the variables.

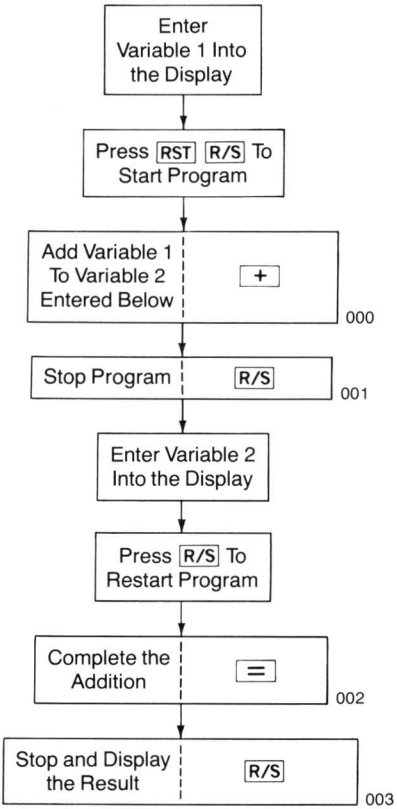
If you want your calculator to find a variable in its memory, place the instruction to recall the variable from the appropriate data register right in your program. For example, recalling a variable stored in data register 1 is performed by the sequence [RCL] **01.**




**Variables in Data Memory Flow Chart**

Let's briefly review what we've accomplished thus far. First, we identified a problem and then considered two methods for entering the variables (one with memory and the other without memory). Third, we developed a simple flow diagram for each method. Notice that a flow diagram is originated by graphically separating the problem into individual steps or actions which solve the problem when performed from top to bottom.

The next important step is to use the flow diagram to help determine the keystrokes required to instruct the calculator to solve the problem. Notice in the two following examples that the [RST] key must be pressed before actually running the program. This assures that the program begins with instruction number 000 when the [R/S] key is used to start the program. The following example shows the keystrokes required to instruct the calculator to look for the variables in its display register.




**Variables in the Display Program**

The centered blocks in each flow diagram explain what *you* must do to run the program once you have keyed the program instructions into program memory. These instructions or keystrokes are found in the right halves of the blocks divided by dotted lines. The numbers outside these blocks are the instruction numbers corresponding to the keystrokes inside the blocks. These keystrokes or program instructions may be keyed into program memory by placing the calculator in the learn mode.

# IV

## Programming Considerations

Here's the procedure for getting a program into your calculator.

1. Set the calculator OFF then ON again to clear your calculator.
2. Press the [LRN] key to enter the learn mode. You will know you are in this mode by a unique display format 000 00.
3. Press each key shown in the flow diagram beginning at the top. Be careful to press only the keys shown. If you make a mistake, start over with step one. Changes in the display are explained below.
4. Press the [LRN] key a second time to exit the learn mode and the unique display disappears leaving a single zero displayed. You are now ready to run the program.

The three digits in the left of the display should change as you enter a program. These three digits show you at what program location or instruction number the *program pointer* is located. The program pointer is an internal device used by the calculator to determine which instruction it should perform next when executing a program. In the learn mode the program pointer simply points to the next unfilled location in the program memory.

Now you and your calculator can try out the program with the variables coming in through the display.

1. Turn the calculator ON.
2. Press the [LRN] key to enter the learn mode.
3. Enter the program by pressing the sequence shown below.

   [+] [R/S] [=] [R/S]

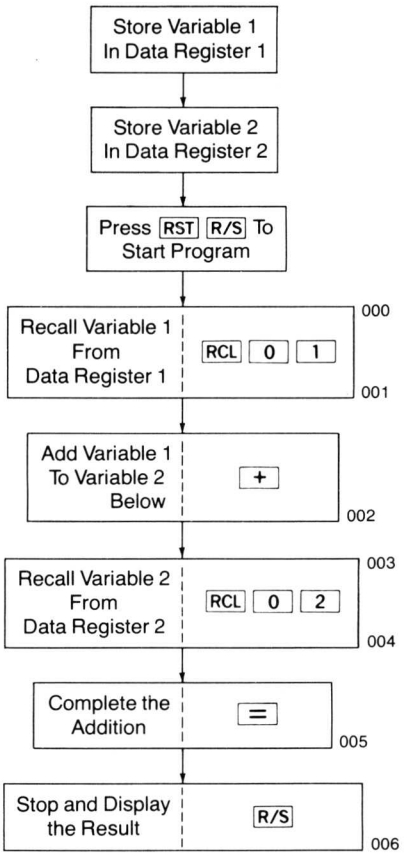
4. Press the [LRN] key to exit from the learn mode.

You have just programmed the calculator. Now solve the problem 227 + 34 = ? by running the program.

1. Press [CLR] to clear any calculations in progress.
2. Press [RST] and enter **227** for variable 1.
3. Press [R/S]. The number **227** remains in the display.
4. Enter **34** for variable 2 and press [R/S] again. The answer **261** is displayed.

Before running any program, it is a safe practice to press the [CLR] key to ensure that there are no pending calculations left to cause erroneous results. Each time these keys are pressed, the program adds the values in data registers 1 and 2 and displays the result. Enter numbers of your own into the data registers and run the program again.

Now let's use data registers to hold our variables and write a new program.




**Variables in Data Memory Program**

Perform the following sequence to enter this program into your calculator.

1. Turn the calculator ON.

2. Press the [LRN] key to enter the learn mode.

3. Enter the program by pressing the following sequence.

[RCL]
[0]
[1]
[+]
[RCL]
[0]
[2]
[=]
[R/S]

4. Press the [LRN] key again to exit from the learn mode.

This program looks for the variables in data registers 01 and 02. Therefore, store 227 in register 01 and 34 in register 02 as follows.

1. To Store 227 in memory 1
   Enter **227**
   Press [STO] **01**

2. To Store 34 in memory 2
   Enter **34**
   Press [STO] **02**

As noted in the flow diagram comments, the only keyboard operation needed to run the program is to press [RST] [R/S]. After pressing these keys, the answer 261 appears in the display and the problem is solved.

Notice that entering the variables from the keyboard into the display took fewer programmed keystrokes or instructions to the calculator, consequently less space was used in program memory. By using the data registers to store the variables, more instructions are placed in program memory, but the calculator computed the result from start to finish with no intermediate stops. Choosing one method over the other depends on your needs.

You should remember that flow diagrams can be very useful, particularly in helping to organize and lay out the approach to solving a particular problem. A flow diagram consists of what is happening while the program is running, and includes not only the instructions or program placed in the calculator's program memory, but also explains what you need to do manually at the keyboard to make the program run, like starting the program and inputting variables. The keystrokes shown are instructions the calculator recognizes and will follow. These keystrokes are stored in the calculator's program memory when the calculator is in the learn mode; essentially they are the program.

# Programming Considerations

# IV

## Mechanics of Programming

The versatile arithmetic language permits both simple and complex programming. Simple programs may be entered, checked, and run with little effort or difficulty. Even though the language is designed to be as straightforward as possible a complex program requires forethought and planning.

If you have done little programming, you will find the following ideas useful. If you are familiar with programming concepts, the ideas will serve as a review and orient you toward calculator programming. You should interpret the following only as a list of suggestions since you will undoubtedly develop your own programming style.

1. **Define the problem very clearly and carefully.** 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** (sometimes called an **algorithm**). Define the operation sequence of the numerical approach you want to use keeping in mind the calculating and programming capabilities of the calculator. (Remember, strictly speaking, calculators do not solve problems, you do. Your calculator carries out your solutions precisely the way you tell it to!)
3. **Develop a flow diagram.** It is often useful to develop drawings that help you visualize the flow of the program. Here, you can picture interactions between various parts of the solution. It may even be possible to simplify the program structure after it is flow charted.
4. **Begin making data register assignments.** Assign data registers to the numerous things you'll be operating on. You'll continue this task throughout the programming process. It is a good idea to never store a quantity in memory without making a written note that the data register in question contains that quantity.
5. **Translate the flow diagram into keystrokes.** The coding forms are provided to help you here. It is useful to list all labels and memory registers in the space indicated on the form. Use the comments column for easy reference to various segments of the program.
6. **Enter the program.** Press [2nd] [CP] [LRN] and key in the complete program from the coding form. When entry is complete, press [LRN] to remove the calculator from the learn mode.
7. **Test the program.** Check out the program using test problems representing as many cases as practical.
8. **Correct errors.** Correct the coding form for any errors discovered while testing the program.
9. **Edit the program.** Place the calculator in the learn mode, complete the required corrections and press [LRN] to return to the keyboard operation.
10. **Retest the program.** Repeat steps 7-9 as needed.
11. **Record the program.** If your calculator has magnetic card capability, record the program on magnetic cards.
12. **Document user instructions.** It's always a good idea to carefully write down step-by-step instructions describing how to use your program. Even the most powerful programs are useless if you don't remember how to use them. Fill out a User Instructions form, detailing information required to run the program.

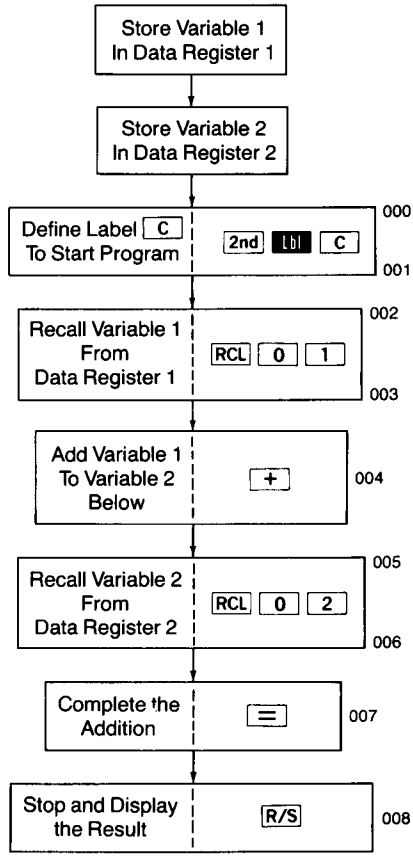
# IV

# Programming Considerations

## Using User-Defined Keys (Labels)

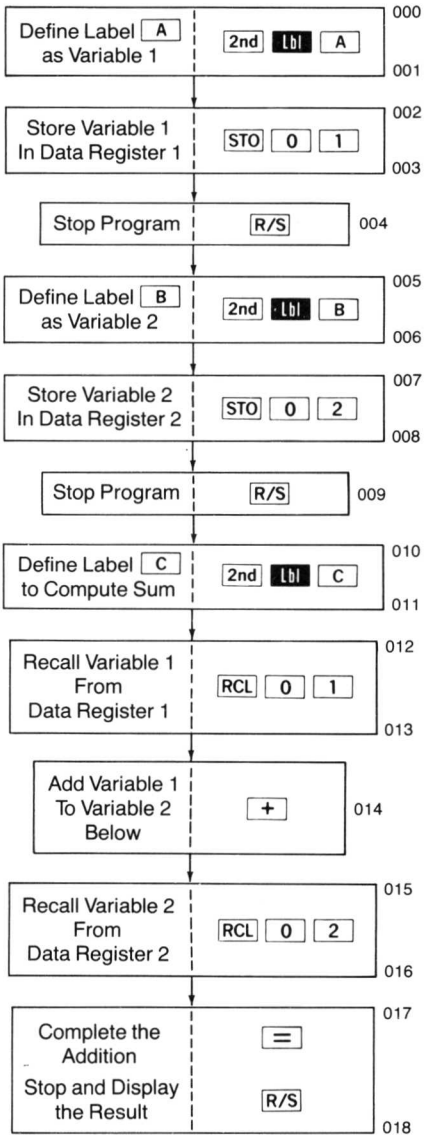
In running the previous sample programs, you used the [RST] and [R/S] keys each time. Since [RST] returns the program pointer to instruction number 000, you may have concluded that every program must start at the beginning of program memory. As you gain programming experience, you will discover that this is not always practical. Your calculator has *user-defined keys* that may be used as *labels* to provide easy access to any location within a program. *Labels* are placed in a program as reference points.

When a user-defined key is placed in a program, pressing this key causes the program pointer to locate the label. The calculator then automatically begins running the program, starting calculations from the first instruction following the label. These keys are [A] through [E] and their second functions [2nd] [A'] through [2nd] [E'] which allow you to identify and access up to ten different reference points (programs or parts of programs). For example, with a minor addition to the first program example, the [RST] [R/S] key sequence used to start the program could simply be [C] or any other user-defined key. Since the [C] key is user-defined, the addition to the program is simply to label the start of the program with [C] using the [2nd] [Lbl] or label key as shown in the following diagram.




Enter the learn mode just as before and key in the keystrokes shown. Exit the learn mode and store 227 in data register 1 and 34 in data register 2. Now press C . The answer 261 is displayed because pressing C tells the calculator to find where Lbl C is located in program memory and then begin executing the instructions or keystrokes following Lbl C .

Now that you have an idea about how user-defined keys work, consider a second addition to the previous program allowing variable 1 to be stored in data register 1 by pressing A and variable 2 to be stored in data register 2 by pressing B . Then, C is again used to obtain the answer.


Define Label A as Variable 1
2nd Lbl A
000
001
Store Variable 1 In Data Register 1
STO 0 1
002
003
Stop Program
R/S
004
Define Label B as Variable 2
2nd Lbl B
005
006
Store Variable 2 In Data Register 2
STO 0 2
007
008
Stop Program
R/S
009
Define Label C to Compute Sum
2nd Lbl C
010
011
Recall Variable 1 From Data Register 1
RCL 0 1
012
013
Add Variable 1 To Variable 2 Below
+
014
Recall Variable 2 From Data Register 2
RCL 0 2
015
016
Complete the Addition
=
Stop and Display the Result
R/S
017
018

Place your calculator in the learn mode and enter the program instructions. Press [LRN] again to exit the learn mode and try out the program. Note that variables 1 and 2 may be entered in either order.

Although this change increases the size of the program it is now much easier to use. The following comparison of these three programs 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** |
|---|---|---|
| Enter **227** | Enter **227** | Enter **227** |
| Press [STO] **01** | Press [STO] **01** | Press [A] |
| Enter **34** | Enter **34** | Enter **34** |
| Press [STO] **02** | Press [STO] **02** | Press [B] |
| Press [RST] [R/S] | Press [C] | Press [C] |
| Display **261** | Display **261** | Display **261** |

Labels may be placed anywhere in a program instruction sequence without altering the meaning of that sequence. They are simply ignored during instruction processing except for the purpose of locating a desired point in program memory and do not affect pending operations. This statement is not intended to mean that a label can interrupt a sequence such as [RCL] **14** where more than one program location is involved in defining a single processing action.

You should include needed labels in your original program design rather than add them as an afterthought. Then key your labels into program memory along with the rest of your code, just as though they were any other instructions.

Of course, even the use of labels does not make practical a program that simply adds two numbers together as the number of keystrokes required for the operation is increased rather than reduced. It should be evident, however, that labels can be used as valuable tools in more sophisticated programs.

## Short Form Addressing

Up till now we have always used a two-digit address to access data registers. That is, recalling a variable stored in data register 1 for example, has been accomplished by using the sequence [RCL] **01.** In some cases, however, leading zeros are not needed to access data registers 0-9. This type of addressing, often called *short form addressing,* may be used *whenever a nonnumeric keystroke immediately follows the register address.*

Example: Store 227 in data register 1 and 34 in data register 2, then recall these values and compute their sum.

| Press | Display | Comments |
|---|---|---|
| **227** [STO] **01** | 227. | Since the next entry is to be a numeric keystroke, the full address must be used. |
| **34** [STO] **2** | 34. | Short form addressing may be used in the |
| [RCL] **1** [+] | 227. | last three occurrences since each |
| [RCL] **2** [=] | 261. | address is followed by a nonnumeric keystroke. |

Observe that when short form addressing is used the instruction is not completed until a nonnumeric key is pressed. That is, just as 227 is not recalled until [+] is pressed, 34 is not stored until [RCL] is pressed.

# Programming Considerations

# IV

## Keying in Your Program

Programming is the technique of determining what your instructions to the calculator are going to be. However, once you have prepared these instructions you need to know how to enter them into the calculator. You have already been exposed to the learn mode; but this section covers it in depth.

Programs are developed through a logical organization of the problem. Although you don't need a calculator to develop the programs, you will want to try each program as it is presented in this manual. Therefore, this section is placed here with the intention of familiarizing you with the learn mode and hopefully assist you in bridging the gap between *writing* a program and *using* a program.

Your calculator can receive program instructions from the keyboard only when it is in the learn mode. Conversely, any keystroke (except the four discussed in *Editing Programs* a few pages later) made when the calculator is in the learn mode is received by the calculator as an instruction. This is an extremely important fact because it means instructions should be entered with care and that keyboard calculations cannot be performed in the learn mode.

If you make a mistake while keying in an instruction, you don't need to start all over reentering the entire sequence of instructions. Your calculator has keys that make it possible to correct a keystroke entered by an erroneously pressed key, to delete instructions, and to add instructions. These keys are discussed in *Editing Programs*.

Following these simple steps should allow you to enter any program.

1. From the keyboard, press [2nd] [CP] to position the program pointer at location 000 and clear all of program memory. This replaces the need to turn the calculator OFF then ON again.
2. Press [LRN] to place the calculator in the learn mode. (Refer to *Displaying the Program* on the opposite page for an explanation of the display format.)
3. Key in your program, not forgetting any necessary [2nd] prefixes.
4. Make sure your program does not exceed the program memory size. If too many instructions are entered, the calculator switches to keyboard control, and the learn mode display format is conspicuously absent.
5. Switch from the learn mode to the keyboard control by pressing [LRN] .
6. Run test problems and correct or edit your program according to the procedures outlined in *Editing Programs*.

# IV

# Programming Considerations

## Displaying the Program

The display format of the learn mode is designed to show you where the program pointer is positioned and the instruction presently found at that location. Turn your calculator on and press [LRN] to enter the learn mode. A unique display consisting of two groups of zeros should appear in the display.


The group of three digits on the left shows you where the program pointer is positioned in program memory. When writing a program, assign each instruction to a location in program memory. This not only allows you to keep track of instructions, but tells the calculator the order in which to complete the instructions as well.

Since your calculator can only understand numbers, each key is assigned a two-digit code number known as a *key code*. The group of digits on the right displays the key code corresponding to the instruction stored in the program memory location indicated on the left. The general rules for coding the keys are:

1. All *numeric keys* are represented by their appropriate number, thus key [7] is coded as "07."

2. All other *first function keys* are assigned key codes relative to their location on the keyboard. The first digit denotes in which of the nine rows (numbered 1-9 from top to bottom) the key is located. The second digit establishes the column location (numbered 1-5 from left to right). In the learn mode, a [$\sqrt{x}$] instruction (row 3, column 4) stored in program memory location 073 would appear as in the display below.

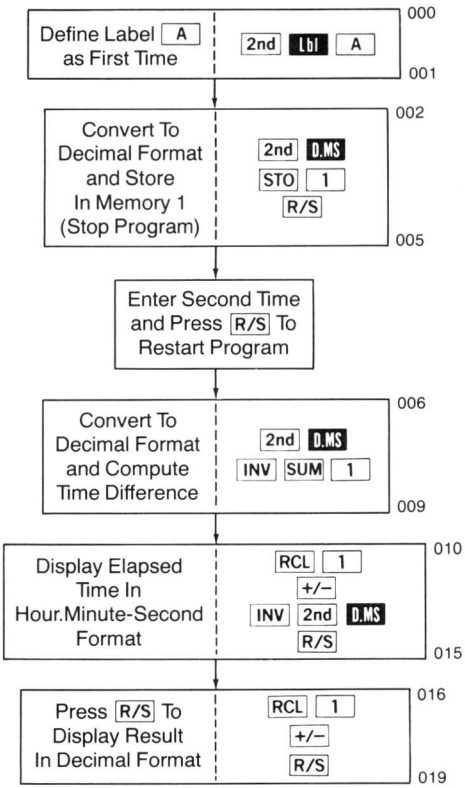

3. *Second functions* are coded by adding five to the column position. The row number is not changed. For example, [2nd] [cos], located above [$\sqrt{x}$] (key code "39"), is coded as "39". [2nd] [tan], however, located above the [$1/x$] key (key code "35"), is represented by "30" rather than "40" as the row number remains the same.

You may use the key code overlay supplied with your calculator to help acquaint yourself with these codes. Key code charts are also found in *Instruction Codes* (Key Codes) in Section V.

In special cases the instructions may be combined and stored in one program location. When this happens, the key codes are also combined or *merged* into a single two-digit code. [RCL] **12** for example, is stored in two program locations. [RCL] is stored in the first location and the sequence [1] [2] is merged and stored in the second. In this example, the key codes are merged into the two-digit code "12". Here, "12" is not interpreted as the user-defined key instruction [B] as any memory operation tells the calculator to translate the next instruction it encounters into a data register address. Other special cases are discussed as they arise.

## Elapsed Time Program

Write a program that may be used to determine the elapsed time between two specified times. You may enter the time in hour.minute-second format (e.g., 3:16:03 = 3.1603) and convert to decimal format for computation using [2nd] [D.MS] . See the discussion of this conversion in *Degree Format Conversion* in Section II.



Let's perform this exercise using the following procedure. First, press [2nd] [CP] to clear program memory and to place the program pointer at location 000 and then enter this program according to the procedure outlined below.

| Press | Display |
|---|---|
| [LRN] | 000 00 |
| [2nd] [Lbl] | 001 00 |
| [A] | 002 00 |
| [2nd] [D.MS] | 003 00 |
| [STO] | 004 00 |
| [1] | *004 00 |
| [R/S] | 006 00 |
| [2nd] [D.MS] | 007 00 |
| [INV] | 008 00 |
| [SUM] | 009 00 |
| [1] | *009 00 |
| [RCL] | 011 00 |
| [1] | *011 00 |
| [+/−] | 013 00 |
| [INV] | 014 00 |
| [2nd] [D.MS] | 015 00 |
| [R/S] | 016 00 |
| [RCL] | 017 00 |
| [1] | *017 00 |
| [+/−] | 019 00 |
| [R/S] | 020 00 |

Since you intitally cleared program memory using [2nd] [CP] all key codes are displayed as 00 as you enter this program. This is because once a program location is filled the program pointer advances to the next location and displays the key code of the instruction stored there — not the instruction just entered.

*These displays do not appear consistent; however, observe that short form addressing is used here. What happens is that the calculator is waiting for the data register address to be completed. In the first occurrence, for example, if [STO] **01** were pressed 005 00 would appear in the display. In the above, however, the nonnumeric keystroke [R/S], tells the calculator that the address stored in 004 is complete and instructs it to store [R/S] in 005. The pointer then automatically advances to 006 as location 005 is filled.

Follow this procedure to verify that you hve entered the program correctly. If you have a PC-100A Printer, just press [RST] [2nd] [List] to obtain this listing.

| Press | Display | Corresponding Keystrokes |
|---|---|---|
| [RST] [LRN] | 000 76 | [2nd] [Lbl] |
| [SST] | 001 11 | [A] |
| [SST] | 002 88 | [2nd] [D.MS] |
| [SST] | 003 42 | [STO] |
| [SST] | 004 01 | [1] |
| [SST] | 005 91 | [R/S] |
| [SST] | 006 88 | [2nd] [D.MS] |
| [SST] | 007 22 | [INV] |
| [SST] | 008 44 | [SUM] |
| [SST] | 009 01 | [1] |
| [SST] | 010 43 | [RCL] |
| [SST] | 011 01 | [1] |
| [SST] | 012 94 | [+/−] |
| [SST] | 013 22 | [INV] |
| [SST] | 014 88 | [2nd] [D.MS] |
| [SST] | 015 91 | [R/S] |
| [SST] | 016 43 | [RCL] |
| [SST] | 017 01 | [1] |
| [SST] | 018 94 | [+/−] |
| [SST] | 019 91 | [R/S] |
| [LRN] | | |

Now that you have correctly programmed your calculator, run this program using 2:15 for the first time and 3:42:54 for the second.

| Enter | Press | Display | Comments |
|---|---|---|---|
| **2.15** | [A] | 2.25 | $t_1$ (H.MMSS)→$t_1$ (H.hh) |
| **3.4254** | [R/S] | 1.2754 | $t_2$(H.MMSS)→$\Delta t$ (H.MMSS) |
| | [R/S] | 1.465 | →$\Delta t$( H.hh) |

The elapsed time is 1 hour, 27 minutes, and 54 seconds, or 1.465 hours.

If in running this program you obtain an output such as 6.396 in the hour.minute-second format interpret this result as 6 hours, 39 minutes, and 60 seconds which is equivalent to 6 hours and 40 minutes.

## Editing Programs

While in the learn mode you have the following capabilities:

1. Display the instruction stored at any program location you choose.
2. Replace an instruction with another.
3. Delete an instruction and close up the hole.
4. Create a space for an additional instruction without destroying previously programmed instructions.
5. Single-step forward or backward through program memory without disturbing its contents.

These features allow you to inspect, correct, and modify a program without having to reenter correct instructions.

The four keys that may be pressed while in the learn mode without being interpreted as a program instruction are [SST], [BST], [Ins], and [Del]. Briefly, [SST] and [BST] allow you to step forward and backward through program memory and examine its contents one location at a time. From the keyboard, [SST] may be used to execute a program one step at a time allowing you to observe the results of each operation. The [2nd] [Ins] instruction causes the current instruction and all following instructions to be advanced one location in program memory while inserting a null instruction at the current location. If an instruction is stored in the last program location it is lost as a result of pressing this key. Pressing [2nd] [Del] causes the instruction at the current program location to be deleted and shifts all following instructions back one location, and fills the last location with a zero.

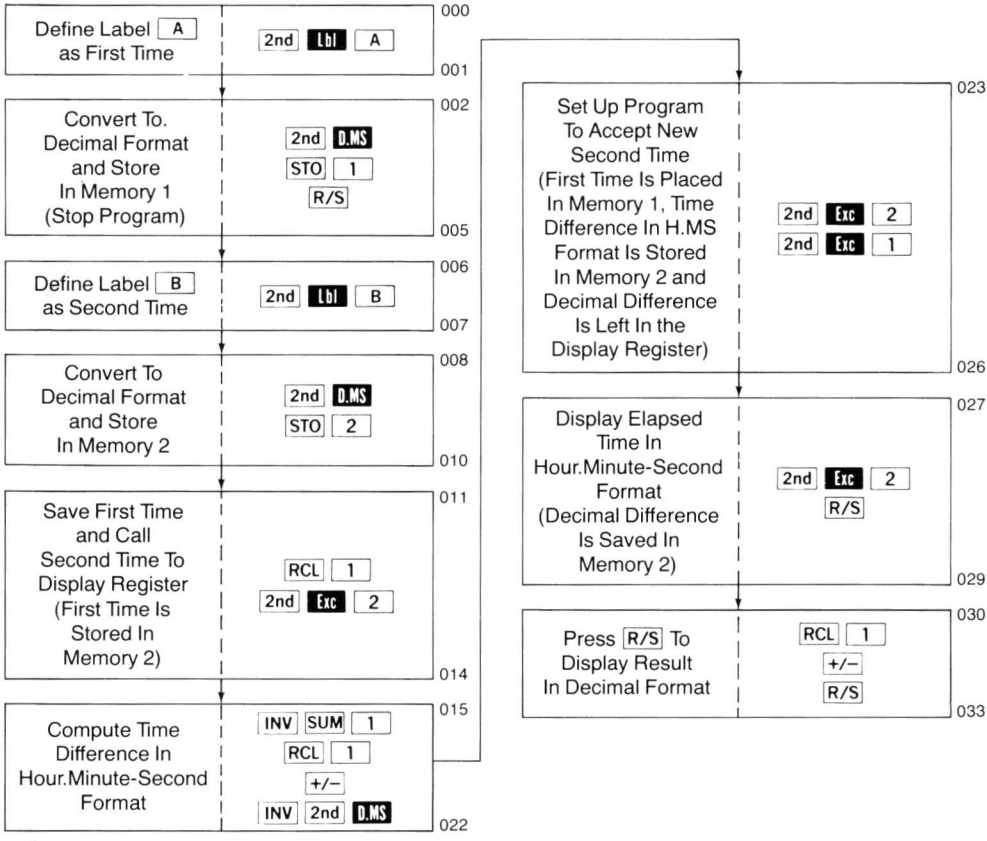
Two additional keys useful in program editing are [RST] (Caution: [RST] performs several functions. See *Basic Program Control Functions* in Section V.) and [GTO]. Pressing [RST] from the keyboard places the program pointer at 000. Pressing [GTO] followed by a three-digit absolute program address or a label, repositions the program pointer to that location in program memory. (Leading zeros may be suppressed in short form addressing.) Pressing [GTO] followed by a label address causes the program pointer to be positioned at the first location following the label in the program. Entering the learn mode following any of the above sequences allows you to examine the contents of program memory at the desired location. **Note that if these keys are pressed while your calculator is in the learn mode they are interpreted as program instructions.**

If you want to change a program sequence, locate the sequence using one of the methods described above and simply cover up the old instructions by entering the new ones or add and delete instructions as needed.

## Improving the Elapsed Time Program

Let's modify the last program so that the second time may be changed without reentering the first time. Make the modification after the original program has been entered rather than keying in the entire program again.

We need to do three things. First, use a label to enter the second time so that this segment of the program may be accessed directly. Second, provide a means of saving the first time so that it may be retrieved after computation. And third, set up the program to accept a new second time after computing the elapsed time.




There is a mistake in the key sequence given above. See if you can find it before reading further.

Make the necessary changes according to the procedure outlined below:

| Press | Display | Comments |
|---|---|---|
| GTO 6 | | Set program pointer to location 006 |
| LRN | 006 88 | Enter learn mode |
| 2nd Ins | 006 00 | Insert label B |
| 2nd Ins | 006 00 | |
| 2nd Lbl | 007 00 | |
| B | 008 88 | |
| SST | 009 22 | Advance program pointer one step |
| 2nd Ins | 009 00 | Insert instructions 009-014 |
| 2nd Ins | 009 00 | |
| STO 2 | 010 00 | |
| 2nd Ins | 011 00 | |
| 2nd Ins | 011 00 | |
| RCL 1 | 012 00 | |
| 2nd Ins | 013 00 | |
| 2nd Ins | 013 00 | |
| 2nd Exc 2 | 014 00 | |
| SST SST SST | 018 43 | Advance program pointer eight steps |
| SST SST SST | 021 22 | |
| SST SST | 023 91 | |
| 2nd Ins | 023 00 | Insert Instructions 023-028 |
| 2nd Ins | 023 00 | |
| 2nd Exc 2 | 024 00 | |
| 2nd Ins | 025 00 | |
| 2nd Ins | 025 00 | |
| 2nd Exc 1 | 026 00 | |
| 2nd Ins | 027 00 | |
| 2nd Ins | 027 00 | |
| 2nd Exc 2 | 028 00 | |
| LRN | | Exit learn mode |



You may use this procedure to verify that you have modified the program correctly.

| Press | Display | Corresponding Keystrokes |
|---|---|---|
| RST LRN | 000 76 | 2nd Lbl |
| SST | 001 11 | A |
| SST | 002 88 | 2nd D.MS |
| SST | 003 42 | STO |
| SST | 004 01 | 1 |
| SST | 005 91 | R/S |
| SST | 006 76 | 2nd Lbl |
| SST | 007 12 | B |
| SST | 008 88 | 2nd D.MS |
| SST | 009 42 | STO |
| SST | 010 02 | 2 |
| SST | 011 43 | RCL |
| SST | 012 01 | 1 |
| SST | 013 48 | 2nd Exc |
| SST | 014 02 | 2 |
| SST | 015 22 | INV |
| SST | 016 44 | SUM |
| SST | 017 01 | 1 |
| SST | 018 43 | RCL |
| SST | 019 01 | 1 |
| SST | 020 94 | +/− |
| SST | 021 22 | INV |
| SST | 022 88 | 2nd D.MS |
| SST | 023 48 | 2nd Exc |
| SST | 024 02 | 2 |
| SST | 025 48 | 2nd Exc |
| SST | 026 01 | 1 |
| SST | 027 48 | 2nd Exc |
| SST | 028 02 | 2 |
| SST | 029 91 | R/S |
| SST | 030 43 | RCL |
| SST | 031 01 | 1 |
| SST | 032 94 | +/− |
| SST | 033 91 | R/S |

Run this program using 1:30 for the first time and 2:13:57 and 2:14:24 for the second times

| Enter | Press | Display | Comments |
|---|---|---|---|
| **1.3** | A | 1.5 | $t_1$ (HH.MMSS)→$t_1$ (HH.hh) |
| **2.1357** | B | 0.4357 | $t_2$ (HH.MMSS)→Δt (HH.MMSS) |
| | R/S | −1.5 | ? |

The example is discontinued here because this last answer is obviously wrong. The output should be the elapsed time in decimal hours; however, it is the negative value of the first time represented in decimal hours. Upon inspecting the flow diagram and the accompanying keystrokes, you can see that the desired output has not been lost. The exchange sequences of steps 023-28 have merely transferred this information to data register 2. Therefore, the problem may be eliminated by changing step 031. This correction can be made by simply replacing the instruction.

| Press | Display |
|---|---|
| GTO 3 1 | |
| LRN | 031 01 |
| 2 | 032 94 |
| LRN | |

Now, run the program again.

| Enter | Press | Display | Comments |
|---|---|---|---|
| **1.3** | A | 1.5 | $t_1$ (HH.MMSS)→$t_1$ (HH.hh) |
| **2.1357** | B | 0.4357 | $t_2$ (HH.MMSS)→Δ t (HH.MMSS) |
| | R/S | 0.7325 | →Δt (HH.hh) |
| **2.1424** | B | 0.4424 | $t_2$ (HH.MMSS)→Δt (HH.MMSS) |
| | R/S | 0.74 | →Δt (HH.hh) |

## Editing with Merged Code

Let's suppose for a moment that the output in the above example had been left in data register 12 instead of 2. If this were the case you would need to store 12 in program location 031. Here, you should be extra careful to ensure that this code is properly merged. (See *Displaying the Program* a few pages back.) One method of entering this code is described below.

| Press | Display |
|---|---|
| [CLR] [GTO] 30 | 0 |
| [LRN] | 030 43 |
| [RCL] | 031 01 |
| [1] [2] | 032 94 |
| [LRN] | 0 |

This procedure requires that the [RCL] instruction be reentered so as to instruct the calculator to automatically merge the data register address and store it in a single program location. Observe that short form addressing may be used for program addresses.

# IV

# Programming Considerations

## Typical Programming Applications

### PROGRAMMING IS PERSONAL

Before proceeding, it is important to understand that programming is very definitely a personal thing. This is to say that two people programming the same problem do not necessarily arrive at the same program instructions, although they may get exactly the same result. This is because we are all individuals, and often approach a problem in different ways. Organization processes can differ as a result of different educational and career backgrounds. An engineer with a great deal of mathematical training would probably need to choose an approach requiring the use of complex mathematical equations, whereas a liberal arts major with less mathematical training may solve his problems using basic arithmetic functions in a different approach. One person may be satisfied to use a set of instructions taking a great deal of program memory space, while another person may prefer to look for ways to condense his program to use the minimum amount. Each of us will want to choose a familiar approach.

Your style should grow as you get into the process of programming. You should even find this learning period adventurous and best of all — fun! Don't be afraid to make mistakes through exploration — your calculator won't mind. Tying up a large-scale computer can cost a lot of money, so beginning programmers are often kept away. Your calculator charges you essentially nothing for its time — so take advantage of this fabulous opportunity and experiment with alternate routes, functions, patterns and anything else you can think of!

### INVESTMENT CALCULATION PROGRAM

What advantages do programmable calculators offer? The programmable calculator is designed to obtain solutions faster and with less chance of making errors through repetitive entries. Now, to program a problem that demonstrates how it saves time.

At one time or another everyone has had a savings account where they received interest on the money in the account. 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 it is left alone for two or more periods and at the end of each period, interest is added to what was in the account at the beginning of that period. Thus 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

Nearly everyone is familiar with but may not be aware of the expression for this concept, which can be stated:

> "The future value of money equals its present value times one plus the interest rate multiplied by itself the number of compounding periods."

Mathematically: $FV = PV \times (1 + i)^n$

Before writing the program, you should logically lay out what is to be done. First the interest rate should be entered into the equation as a decimal. Let the calculator do this by dividing the interest rate by 100 after it is entered. Also, 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. Incorporating these changes into the future value equation it may be rewritten as:

$$FV = PV \times \left[ 1 + \left( \frac{i}{100} \div c \right) \right]^{cn}$$

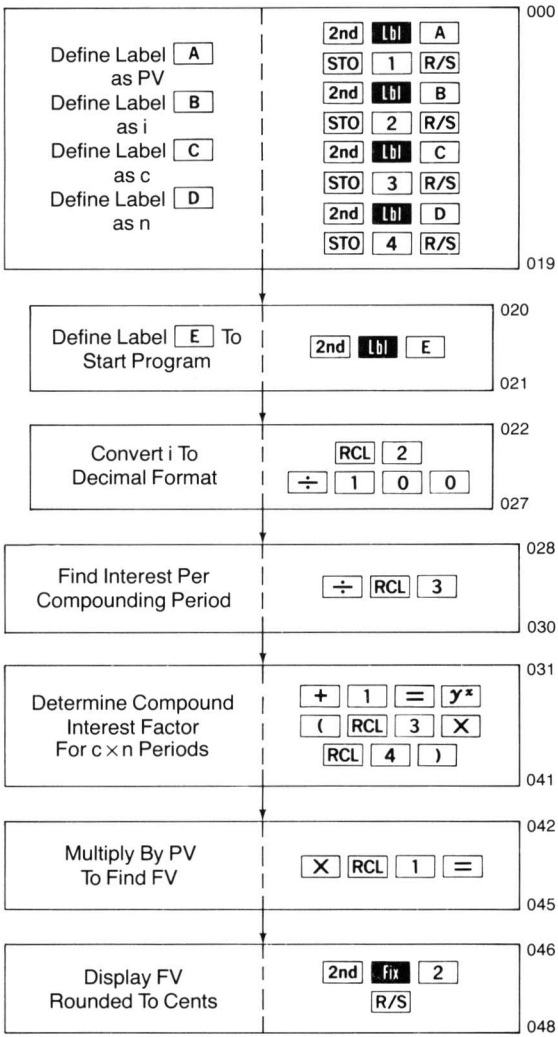
The variables used above are:

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

**Note : In Europe, it is usual to quote annual effective rates for compound interest. These are not simple multiples of the coumpounding - period interest rate but are themselves governed by the compound interest equation. However, for the sake of simplicity we have omitted the European example at this stage.**

Now you should decide whether to enter the variables into the display as each is called for or place them in memory to be recalled as needed. In this example they are placed in memory. This allows the variables to be entered individually making 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. This is the modification that was required by the elapsed time program example.

The approach has been decided, the equation structured to reflect the variables desired, and it has been determined how to handle the variables. Now diagram the approach and write the program.


Define Label A as PV
Define Label B as i
Define Label C as c
Define Label D as n
2nd Lbl A
STO 1 R/S
2nd Lbl B
STO 2 R/S
2nd Lbl C
STO 3 R/S
2nd Lbl D
STO 4 R/S
000
019
Define Label E To Start Program
2nd Lbl E
020
021
Convert i To Decimal Format
RCL 2
÷ 1 0 0
022
027
Find Interest Per Compounding Period
÷ RCL 3
028
030
Determine Compound Interest Factor For c × n Periods
+ 1 = yˣ
( RCL 3 ×
RCL 4 )
031
041
Multiply By PV To Find FV
× RCL 1 =
042
045
Display FV Rounded To Cents
2nd fix 2
R/S
046
048

| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Counter | | 2nd CF | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Investment Calculation Program | | | |
| 4 | Exit Learn Mode | | LRN | 0 |
| 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 | Computer Future Value | | E | FV |
| | Variables May Be Entered In Any Order — There Is No Need to Reenter Variables That Do Not Change For New Problems | | | |

| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 42 | STO |
| 003 01 | 1 |
| 004 91 | R/S |
| 005 76 | 2nd Lbl |
| 006 12 | B |
| 007 42 | STO |
| 008 02 | 2 |
| 009 91 | R/S |
| 010 76 | 2nd Lbl |
| 011 13 | C |
| 012 42 | STO |
| 013 03 | 3 |
| 014 91 | R/S |
| 015 76 | 2nd Lbl |
| 016 14 | D |
| 017 42 | STO |
| 018 04 | 4 |
| 019 91 | R/S |
| 020 76 | 2nd Lbl |
| 021 15 | E |
| 022 43 | RCL |
| 023 02 | 2 |
| 024 55 | ÷ |
| 025 01 | 1 |
| 026 00 | 0 |
| 027 00 | 0 |
| 028 55 | ÷ |
| 029 43 | RCL |
| 030 03 | 3 |
| 031 85 | + |
| 032 01 | 1 |
| 033 95 | = |
| 034 45 | $y^x$ |
| 035 53 | ( |
| 036 43 | RCL |
| 037 03 | 3 |
| 038 65 | × |
| 039 43 | RCL |
| 040 04 | 4 |
| 041 54 | ) |
| 042 65 | × |
| 043 43 | RCL |
| 044 01 | 1 |
| 045 95 | = |
| 046 58 | 2nd Fix |
| 047 02 | 2 |
| 048 91 | R/S |



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

| Enter | 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 data registers primarily for storing and recalling variables. However, the calculator can add to, subract from, multiply and divide the variables previously stored in data registers without recalling them. Using the memory in this fashion is often referred to as memory arithmetic. The value of memory arithmetic is demonstrated by the sample program below. Also note that an extremely useful program can be developed using only simple arithmetic, further emphasizing the fact that programmable calculators are ideal and easy to use in solving any type of program — not just ones involving complex mathematics.

Assume the normal purchase order received in a business is comprised of like items selling at various prices. 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 must total the line item qualities and divide the sum into the total order price. This process is not difficult, but it is time consuming.

| 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 | |

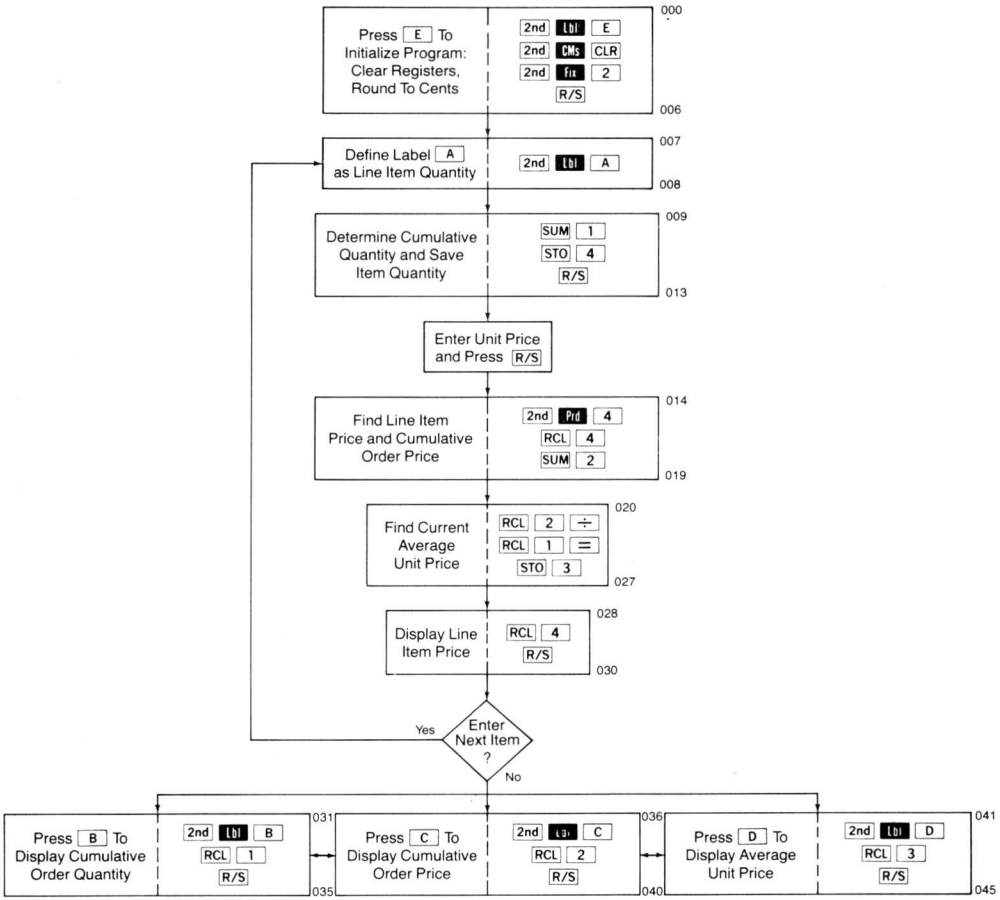
A glance at the order immediately tells one to multiply, add, and divide. The key is how to organize the problem and what to instruct the calculator to do.

If you need a program that can handle an unlimited number of line items, choose another approach. First, decide how to enter your variables. In the example below the variables are entered through the display while the program is running rather than storing them in data registers to be recalled at a later point in the program. If you use memory arithmetic to calculate the cumulative totals, each set of data is used only once and does not need to be permanently stored in data registers. To save time lost by displaying intermediate data, the cumulative order quantity is stored in $R_1$ (data register 1 is denoted by $R_1$, data register 2 by $R_2$, etc.), the cumulative order price is stored in $R_2$, and the current average unit price is stored in $R_3$. The sample program is designed to display the line item price of a given line item after you enter the appropriate quantity and unit price; 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 $R_1$ and $R_2$ are to be sum instructions, the program should be equipped with an initialization routine which zeros these data registers.

With this example, the importance of organizing the approach should be apparent. The first approach would have limited the capability of the program by fixing the number of line items that could be handled for any one order; the second approach allows an order to have an unlimited number of line items.

Now flowchart the problem and determine the keystrokes needed in the program.



| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Pointer | | 2nd CP | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Pricing Control Program | | | |
| 4 | Exit Learn Mode | | LRN | 0 |
| 5 | Initialize Program | | A | 00.00 |
| 6 | Enter Line Item Quantity | Quantity | B | 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 | | C | Order Quantity |
| | Cumulative Cost | | D | Order Price |
| | Average Unit Price | | E | Average Price |

| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | E |
| 002 47 | 2nd CMs |
| 003 25 | CLR |
| 004 58 | 2nd Fix |
| 005 02 | 2 |
| 006 91 | R/S |
| 007 76 | 2nd Lbl |
| 008 11 | A |
| 009 44 | SUM |
| 010 01 | 1 |
| 011 42 | STO |
| 012 04 | 4 |
| 013 91 | R/S |
| 014 49 | 2nd Prd |
| 015 04 | 4 |
| 016 43 | RCL |
| 017 04 | 4 |
| 018 44 | SUM |
| 019 02 | 2 |
| 020 43 | RCL |
| 021 02 | 2 |
| 022 55 | ÷ |
| 023 43 | RCL |

| Location and Key Code | Key Sequence |
|---|---|
| 024 01 | 1 |
| 025 95 | = |
| 026 42 | STO |
| 027 03 | 3 |
| 028 43 | RCL |
| 029 04 | 4 |
| 030 91 | R/S |
| 031 76 | 2nd Lbl |
| 032 12 | B |
| 033 43 | RCL |
| 034 01 | 1 |
| 035 91 | R/S |
| 036 76 | 2nd Lbl |
| 037 13 | C |
| 038 43 | RCL |
| 039 02 | 2 |
| 040 91 | R/S |
| 041 76 | 2nd Lbl |
| 042 14 | D |
| 043 43 | RCL |
| 044 03 | 3 |
| 045 91 | R/S |

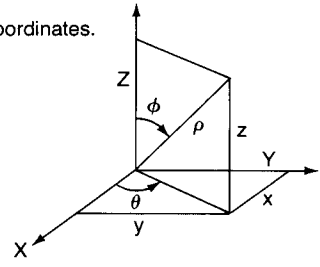
Now let's run the program using the data given earlier.

| Enter | Press | Display | Comments |
|---|---|---|---|
| | A | 0.00 | Initialize |
| 100 | B | 100.00 | Quantity A |
| .25 | R/S | 25.00 | Unit Price A |
| | | | Line Item Price |
| 200 | B | 200.00 | Quantity B |
| .15 | R/S | 30.00 | Unit Price B |
| | | | Line Item Price |
| 50 | B | 50.00 | Quantity C |
| .35 | R/S | 17.50 | Unit Price C |
| | | | Line Item Price |
| 150 | B | 150.00 | Quantity D |
| .4 | R/S | 60.00 | Unit Price D |
| | | | Line Item Price |
| 300 | B | 300.00 | Quantity E |
| .1 | R/S | 30.00 | Unit Price E |
| | | | Line Item Price |
| | C | 800.00 | Total Order Quantity |
| | D | 162.50 | Total Order Price |
| | E | 0.20 | Avg. Unit Price (Rounded) |
| | INV 2nd Fix | 0.203125 | Avg. Unit Price (Exact) |

## SPHERICAL COORDINATES PROGRAM

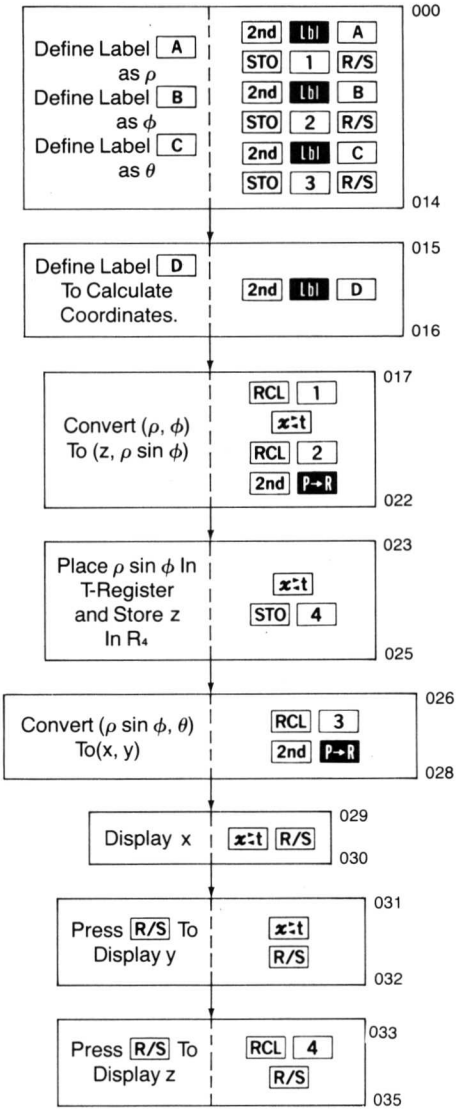
Write a program to convert from spherical to rectangular coordinates.

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




Your calculator has a built-in function, [2nd] [P→R] that is preprogrammed to convert from polar to rectangular coordinates. (See *Conversions* in Section V.) This function could be very useful here.

The easiest way to enter the spherical coordinates is to simply store $\rho$, $\phi$, and $\theta$ in data registers $R_1$, $R_2$, and $R_3$ respectively. Instructing the calculator to place $\rho$ in the T-register and $\phi$ in the display register, allows z to be found by converting to rectangular coordinates using [2nd] [P→R]. This conversion places $\rho \sin \phi$ in the display register and $\rho \cos \theta$ (=z) in the T-register. Storing Z in $R_4$ for safekeeping and placing $\rho \sin \phi$ in the T-register, it is possible to use this conversion again to find x and y after recalling $\theta$ to the display register. The program should be designed so as to display x, y, and z in the given order by using the [R/S] key.

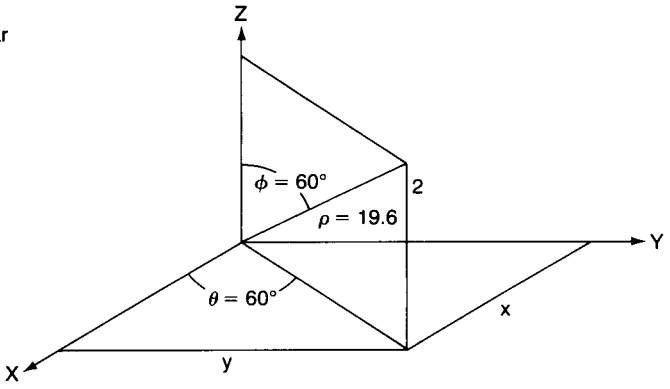

Define Label A as ρ
Define Label B as ϕ
Define Label C as θ
2nd Lbl A
STO 1 R/S
2nd Lbl B
STO 2 R/S
2nd Lbl C
STO 3 R/S
000
014
Define Label D To Calculate Coordinates.
2nd Lbl D
015
016
Convert (ρ, ϕ) To (z, ρ sin ϕ)
RCL 1
x⇄t
RCL 2
2nd P→R
017
022
Place ρ sin ϕ In T-Register and Store z In R4
x⇄t
STO 4
023
025
Convert (ρ sin ϕ, θ) To(x, y)
RCL 3
2nd P→R
026
028
Display x
x⇄t R/S
029
030
Press R/S To Display y
x⇄t
R/S
031
032
Press R/S To Display z
RCL 4
R/S
033
035

| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Pointer | | 2nd CP | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Program | | | |
| 4 | Exit Learn Mode | | LRN | |
| 5 | Enter $\rho$ | $\rho$ | A | $\rho$ |
| 6 | Enter $\phi$ | $\phi$ | B | $\phi$ |
| 7 | Enter $\theta$ | $\theta$ | C | $\theta$ |
| 8A | Compute Coordinates and Display x | | D | x |
| 8B | Display y | | R/S | y |
| 8C | Display z | | R/S | z |

| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 42 | STO |
| 003 01 | 1 |
| 004 91 | R/S |
| 005 76 | 2nd Lbl |
| 006 12 | B |
| 007 42 | STO |
| 008 02 | 2 |
| 009 91 | R/S |
| 010 76 | 2nd Lbl |
| 011 13 | C |
| 012 42 | STO |
| 013 03 | 3 |
| 041 91 | R/S |
| 015 76 | 2nd Lbl |
| 016 14 | D |
| 017 43 | RCL |
| 018 01 | 1 |

| Location and Key Code | Key Sequence |
|---|---|
| 019 32 | x⇄t |
| 020 43 | RCL |
| 021 02 | 2 |
| 022 37 | 2nd P→R |
| 023 32 | x⇄t |
| 024 42 | STO |
| 025 04 | 4 |
| 026 42 | RCL |
| 027 03 | 3 |
| 028 37 | 2nd P→R |
| 029 32 | x⇄t |
| 030 91 | R/S |
| 031 32 | x⇄t |
| 032 91 | R/S |
| 033 43 | RCL |
| 034 04 | 4 |
| 035 91 | R/S |

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




| Enter | Press | Display | Comments |
|---|---|---|---|
| | 2nd Deg | | Place calculator in degree mode. |
| **19.6** | A | 19.6 | $\rho$ |
| **60** | B | 60 | $\phi$ |
| **60** | C | 60 | $\theta$ |
| | D | 8.487048957 | x |
| | R/S | 14.7 | y |
| | R/S | 9.8 | z |

# ADVANCED PROGRAMMING

## More About Labels

As you remember, the user-defined keys (A-E, A'-E') are designed for use as labels. Once a program segment is labeled with one of these keys, pressing it from the keyboard sends the program pointer to that part of the program and the program starts immediately.

In addition to the user-defined keys, you can use almost any first or second function on the calculator as a label. For instance, [2nd] [sin], [$x^2$], [=], [CLR], [2nd] [Pgm], [EE], [2nd] [Fix] and others can be labels. These are called *Common Labels*. Only the following keys *cannot* be used as labels.

[Lbl] [2nd] [SST]
[Ins] [LRN] [BST]
[Del] [Ind] **numbers**

The only difference between the common labels and the user-defined labels is that pressing a common label from the keyboard cannot start program execution. Even though you have a program segment labeled $x^2$, for example, pressing [$x^2$] from the keyboard simply squares the displayed value. The keyboard sequence [GTO] [$x^2$] [R/S] does cause the program to start running at label $x^2$. Nonetheless, you now have over 60 more labels to work with.

Common labels can be used anywhere in a program as can the user-defined labels. However, you must not split obvious instruction clusters like [STO] **12** or [2nd] [Fix] **6** or [INV] [2nd] [sin]. Throughout the remainder of this section we'll primarily discuss user-defined keys as labels because of their versatility.

## Transfer Instructions

There are several important instructions that further increase the programming capabilities of your calculator. They allow you extra flexibility of control over the order in which your program instructions are executed. These new program controls are called *Transfer Instructions* or just simply *transfers*. They can divert the normal "top" flow of a program by jumping to some other location. Basically there are two types of transfers, termed *unconditional* and *conditional*.

Unconditional transfer instructions immediately branch to wherever the program asks, unconditionally. Unconditional transfers are independent of all calculations. A conditional transfer instruction on the other hand tests some value and transfers to a location other than the one next in line if that value fulfills the conditions of the test.

## Unconditional Transfers

[RST], [GTO] and [SBR] are called unconditional transfer instructions. [RST] automatically positions the program pointer at location 000. [GTO] and [SBR] unconditionally place the pointer at the location *you* specify. Note also that [RST] performs additional functions that you should be aware of. (See *Basic Program Control Functions* on page V-43.)

### THE GO TO INSTRUCTION — [GTO]

Back in *Editing Programs* you learned how to use the Go To instruction from the keyboard. It's just the same when running a program, [GTO] followed by an absolute address or a label causes the program pointer to go immediately to that location. Processing continues at the new location.

Short form addressing can be used with absolute addresses (program memory addresses). In the learn mode, for example, pressing [GTO] **9** is the same as [GTO] **009** *if and only if the next keystroke is not a number.* Key in the following little program that counts.

| Press | Display | Comments |
|---|---|---|
| [CLR] [2nd] [CP] | 0. | Prepare for program |
| [GTO] **9** [LRN] | 009 00 | Go To location 009 and enter learn mode |
| [+] | 010 00 | Key in program |
| **1** | 011 00 | |
| [=] | 012 00 | |
| [2nd] [Pause] | 013 00 | |
| [GTO] | 014 00 | |
| **9** | 014 00 | Location number did not advance, waiting for rest of address |
| [LRN] | 0. | Exit learn mode |
| [LRN] | 016 00 | Program pointer advanced when first [LRN] signaled the end of the incoming address |

To run this program, press [GTO] **9** [R/S].

In program memory, absolute addresses are stored in two program locations. For example, the sequence [GTO] **136** occupies 3 program locations. The first location contains the [GTO] instruction while the address is stored in the next two. The hundreds digit of the address goes into the first of these locations and the last two digits are merged to occupy the next. The resulting key code sequence is 61 01 36 in program memory. This compact way of doing things is called "merging" and is completely automatic within the calculator.

Try the following exercise on your calculator.

| Press | Display | Comments |
|---|---|---|
| [2nd] [CP] [CLR] | 0. | Clear program memory and display |
| [GTO] **136** | 0. | Sends the program pointer to location 136 |
| [LRN] | 136 00 | |
| [R/S] | 137 00 | Store [R/S] in location 136 |
| [LRN] [RST] [LRN] | 000 00 | Return to 000 |
| [GTO] | 001 00 | Store [GTO] in 000 |
| **1** | 001 00 | |
| **3** | 001 00 | |
| **6** | 003 00 | Memory waits for all 3 digits before storing address |
| [BST] | 002 36 | |
| [BST] | 001 01 | Address stored as explained |
| [BST] | 000 61 | |
| [LRN] | 0. | |
| [R/S] | 0. | Execute program |
| [LRN] | 137 00 | Transfer is instantly made to location 136 where the [R/S] stored there is executed, leaving the program pointer at location 137. |

The same thing would work for a label. Pressing [GTO] [$x^2$], for example, from the keyboard sends the program pointer to label $x^2$ and awaits further instructions. If you ask the calculator to find a label that doesn't exist, the display flashes its current value.

You can see here how the Go To instruction works when used from the keyboard or in a program. The other transfers operate much the same way. As soon as they are pressed from the keyboard or encountered in a program, they transfer to the requested program location.

# Programming Considerations

# IV

## SUBROUTINES

As you begin writing more and more programs, you'll often find sequences of calculations that are performed repeatedly. These are called *Subroutines.* Subroutines give you the capability to define a "subprocess" or sequence of keystrokes that have a unique purpose. These you can label and reference at any time from anywhere in your program almost as easily as if a key on the keyboard was devoted to it. Once a subroutine has completed its purpose, the program pointer repositions itself to the first program location following the point where you began using it. When you use a subroutine — it is often said that you "call" it — you are telling the machine to run a whole sequence of steps with a single subroutine instruction.

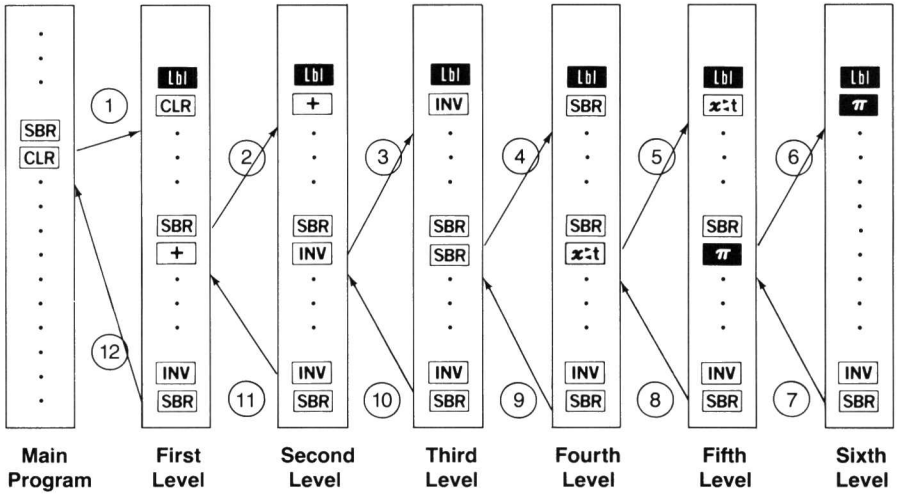
It's a good programming practice to write your programs so that they can be used as subroutines. Now, they can be used by other programs without having to be modified. You may do this by simply using [INV] [SBR] to halt program execution instead of [R/S]. The remaining programs in this section are written using this technique.

## THE SUBROUTINE INSTRUCTION — [SBR]

The subroutine instruction is a Go To that has been modified in two ways — from the keyboard, it can start program execution and in a program, it remembers where it transferred from. From the keyboard, pressing [SBR] **136** sends the program pointer instantly to location 136 and program execution automatically starts at that point. It is exactly the same as pressing [GTO] **136** [R/S]. The same thing happens when you press [SBR] [$x^2$], the program starts running at label [$x^2$], wherever it may be.

If [SBR] **136** had been keyed into locations 000-002 as [GTO] **136** was in the previous example, it would have been executed the same way. But, when executed, program location 003 is stored in the *Subroutine Return Register.* Now, if there is a calculation sequence beginning at 136 that you have ended with [INV] [SBR], processing would look at the subroutine return register, find 003, and bounce back to location 003 where processing continues. To get processing to go back where it came from, simply end the subroutine with [INV] [SBR]. What actually has happened is that you have transferred to a subroutine at 136 and return is automatic when the [INV] [SBR] instruction is encountered.

Actually, as many as six return addresses can be stored in the subroutine return register at any one time. This means that a subroutine can contain and use or "call" a subroutine that can also call a subroutine, etc., — up to six times. This tremendous capability is shown graphically.




If the sequence of steps shown above was written as part of a program, processing would flow as numbered above. Note that [INV] [SBR] ends each subroutine, instructing the calculator to go to the subroutine return register and retrieve the address that was stored last and transfer there. Processing usually ends up back in the *main program* — the one that started calling subroutines. By the way, [INV] [SBR] is merged in program memory to occupy only one location as key code 92. This code doesn't match the row/column key designations.

When a program part is labeled with a user-defined key, that part can be executed from the keyboard simply by pressing the applicable label as we have seen. The same thing happens when one of these keys is encountered in a program — the program pointer goes to that label and processing continues. These user-defined keys have an automatic [SBR] instruction built in. So, if you label a program part with a user-defined key and end it with [INV] [SBR], that part is processed just as though you had called it with the [SBR] instruction.

## ACCESSING OR CALLING SUBROUTINES

To clarify the definition, a subroutine is a segment of a program designed for a specific purpose — to be written once, but used repeatedly. All subroutines must end with [INV] [SBR] to instruct processing to return to the sequence that called it.

There are three methods of calling subroutines.

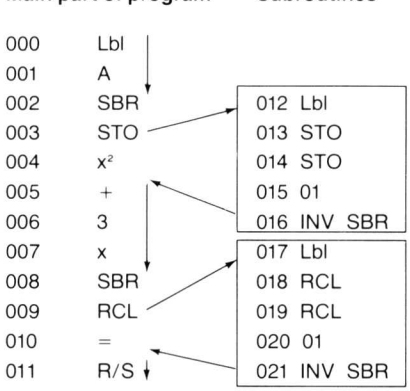
- Absolute address, [SBR] **136**
- Common label, [SBR] [$x^2$]
- User-defined label, [A]

Labeling subroutines adds clarity and simplicity to the program instructions. A label descriptive of the purpose of the subroutine can even be used. You should choose your labels well and record the meaning of each. Labeled subroutines can be placed anywhere in program memory because, when called, a label can be found regardless of where it is. Also, a subroutine that is labeled is not affected by insert and delete instructions that are performed ahead of the subroutine in program memory.

To have a program evaluate $x^2 + 3x$ for incoming x values, simply key in the following.

[STO]
[1]
[$x^2$]
[+]
[3]
[×]
[RCL]
[1]
[=]
[R/S]

For demonstration, let's let subroutines do our storing and recalling. The program looks like this.




Enter any x value and press [A]. The sequence of processing is charted by the arrows. Note that a user-defined label is used to start the program because that's the easiest way to do it. The subroutine names were chosen to identify their respective segments, but other labels could have been used just as well.

## THINGS TO WATCH OUT FOR IN SUBROUTINES

Two instructions that should be used very carefully in subroutines are Reset [RST] and Equals [=]. Also you need to be sure that the subroutine return register is cleared before the start of a new problem.

The reset instruction, among its other functions, automatically clears the subroutine return register. If you do need to transfer to location 000 (the primary function of [RST]) in a subroutine, use [GTO] **000** or a label if there is one at 000.

The equals instruction completes *all* pending operations. If used in a subroutine, the pending operations not only of the subroutine, but those of the main program are completed as well.

Consider the following program segment to evaluate 4 + (1 + 2) × 3.

| | |
|---|---|
| 4 | |
| + | |
| SBR $x^2$ | |
| × | |
| 3 | |
| = | |
| R/S | |
| 2nd Lbl $x^2$ | |
| 1 | |
| + | |
| 2 | |
| = | The equals here in subroutine $x^2$ not only completes 1 + 2, but |
| INV SBR | also the 4 + before returning to the × 3. The resulting answer is now 21 where it should be 13. |

This program can be easily modified to correctly handle the problem by using parentheses to evaluate the subroutine.

| |
|---|
| 4 |
| + |
| SBR $x^2$ |
| × |
| 3 |
| = |
| R/S |
| 2nd Lbl $x^2$ |
| ( |
| 1 |
| + |
| 2 |
| ) |
| INV SBR |

This sequence yields the correct answer, 13.

Beginning each subroutine with [(] and placing [)] right before the [INV] [SBR] is a good habit to develop. It takes an extra keystroke as opposed to using [=], but can save you much misery down the road. The primary advantage being that parentheses affect no pending operations other than those contained within that parenthesis set.

Avoiding the equals [=] instruction in such cases should impose no hardship as parentheses are designed to selectively complete expressions like this. However, there are some things you need to know to use the current display register value in the subroutine.

Whenever a subroutine requires repeated access to the display register contents at the time of the call, you should store the variable in a data register prior to performing calculations and recall it whenever it is needed. If the contents of the display register are needed only to begin computation, the [CE] key may be used as a trick to pull this value inside the parentheses. This trick works the same in a program as it does from the keyboard.

**Press: 2.18 [×] [(] [CE] [+] 6 [)] [=]** **Display: 17.8324**

In the above sequence the [CE] key pulls 2.18 inside the parentheses and enables the calculator to evaluate $2.18 \times (2.18 + 6) = 17.8324$.

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 routine. In such situations return of control remains pending as the subroutine return register has not been cleared. Unless you turn the calculator off, use [RST] or clear the return register using the [2nd] [CP] instruction, 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 ruining future problems, you should use the [RST] instruction to clear the subroutine return register. You may do this manually, but it is preferable, whenever possible, to include [RST] at a proper point in the program.

# Programming Considerations

# IV

## LIBRARY PROGRAMS AS SUBROUTINES

You may often find it quite useful to extend the power of a program you write through effective use of library programs. The same instructions used to access these programs from the keyboard can be built right into your program. The library programs simply become subroutines of your program in program memory. Thousands of instructions worth of subroutines are stored in each of the library modules. Each program of a library is a subroutine. They do not use [=] or [RST] and are evaluated with parentheses and end in [INV] [SBR].

From the keyboard, pressing [2nd] [Pgm] **mm** relocates the program pointer to the library program numbered mm. Upon completion of a library program segment, the program pointer remains in that program.

In a program, the function of [Pgm] is similar to that of the [SBR] instruction. The only difference is that [2nd] [Pgm] **mm** tells the calculator to look for the subroutine in library program **mm** rather than in program memory of the calculator itself. Once the library routine is completed, the program pointer returns to the point of call in program memory and normal processing continues. The two-digit program number is merged and stored in a single program location.

If a segment of a library routine is identified by a user-defined key such as [A], the program sequence required to execute that routine is [2nd] [Pgm] **mm** [A]. If a label such as [tan] is used, the sequence becomes [2nd] [Pgm] **mm** [SBR] [2nd] [tan]. Note that following [2nd] [Pgm] **mm** with anything other than [SBR] or a user-defined key is not a valid key sequence and can produce unwanted results.

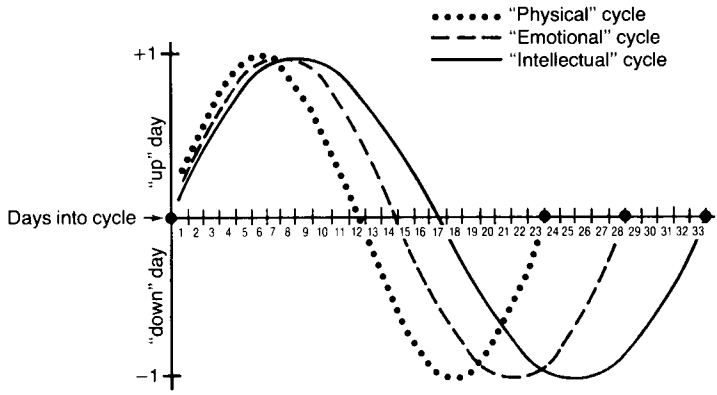
## BIORHYTHM PROGRAM

Let's examine an illustrative example of subroutines in action in an interesting program situation.

The theory of biorhythms states that there are three cycles to your life that started on the day you were born:

1. The Physical cycle — 23 days long
2. The Emotional cycle — 28 days long
3. The Intellectual cycle — 33 days long.

The first half of each cycle is said to contain your "up days" while the lower half represents "down days."

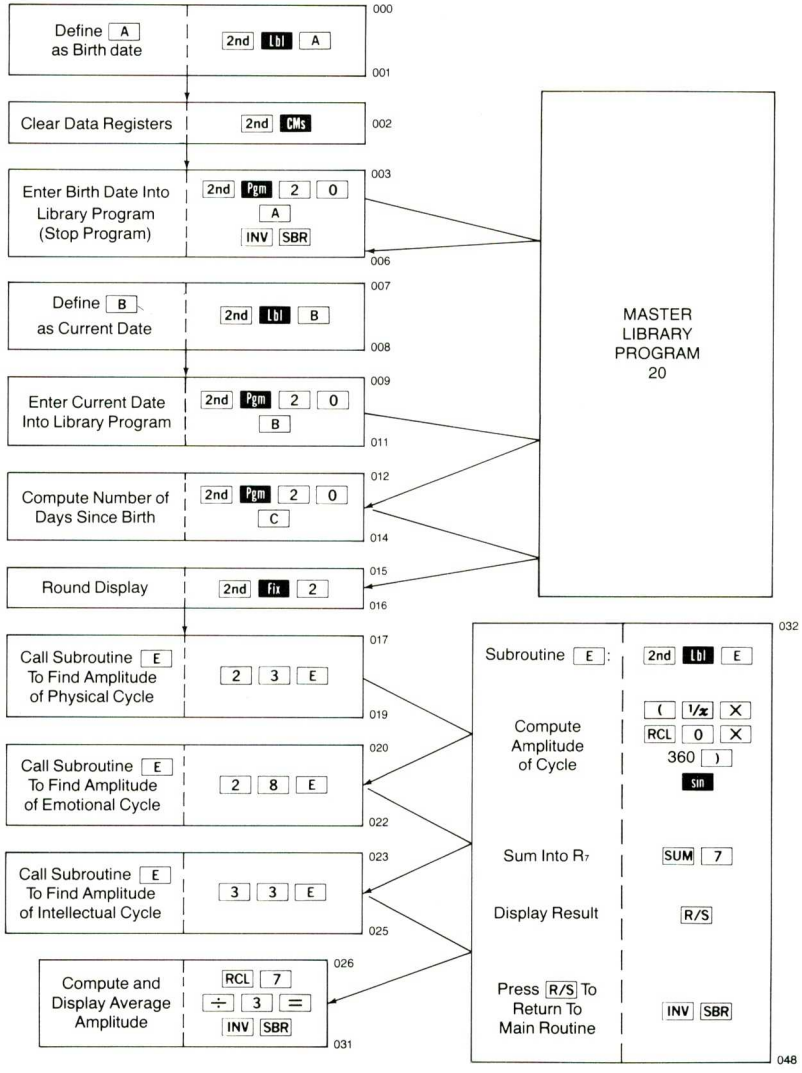



The amplitudes of these biorhythm cycles on a given day may be expressed as a value between −1 and 1 using the following equation.

$$\text{Amplitude} = \sin\left(360 \times \frac{\text{Number of Days Since Birth}}{\text{Number of Days in Cycle}}\right).$$

Now let's write a program designed to determine the amplitudes of a person's biorhythmic cycles. Since each cycle uses the same equation, a subroutine may be used to simplify your program. Also you can use program 20 of the Master Library as a subroutine to compute the number of days since birth.

In the sample program below, the amplitude of each cycle is computed using the subroutine labeled [E] since the only difference between calculations is the number of days in the cycles. We can use [R/S] to stop the program and display the amplitudes of the physical, emotional, and intellectual cycles in that order. [INV] [SBR] is not used here because it would return program control to the main program rather than stop execution (see *The Subroutine Instruction* — [SBR] a few pages back). The final result of the program is the average of these three values. All results are rounded to two decimal places for display.




**Biorhythm Program**

| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 47 | 2nd CMs |
| 003 36 | 2nd Pgm |
| 004 20 | 2 0 |
| 005 11 | A |
| 006 92 | INV SBR |
| 007 76 | 2nd Lbl |
| 008 12 | B |
| 009 36 | 2nd Pgm |
| 010 20 | 2 0 |
| 011 12 | B |
| 012 36 | 2nd Pgm |
| 013 20 | 2 0 |
| 014 13 | C |
| 015 42 | STO |
| 016 00 | 0 |
| 017 58 | 2nd Fix |
| 018 02 | 2 |
| 019 02 | 2 |
| 020 03 | 3 |
| 021 15 | E |
| 022 02 | 2 |
| 023 08 | 8 |
| 024 15 | E |
| 025 03 | 3 |
| 026 03 | 3 |

| Location and Key Code | Key Sequence |
|---|---|
| 027 15 | E |
| 028 43 | RCL |
| 029 07 | 7 |
| 030 55 | ÷ |
| 031 03 | 3 |
| 032 95 | = |
| 033 92 | INV SBR |
| 034 76 | 2nd Lbl |
| 035 15 | E |
| 036 53 | ( |
| 037 35 | 1/x |
| 038 65 | × |
| 039 43 | RCL |
| 040 00 | 0 |
| 041 65 | × |
| 042 03 | 3 |
| 043 06 | 6 |
| 044 00 | 0 |
| 045 54 | ) |
| 046 38 | 2nd sin |
| 047 44 | SUM |
| 048 07 | 7 |
| 049 91 | R/S |
| 050 92 | INV SBR |

**Biorhythm Program**

| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Pointer | | 2nd CP | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Biorhythm Program | | | |
| 4 | Exit Learn Mode | | LRN | |
| 5 | Enter Birth Date | MMDD.YYYY | A | 0. |
| 6 | Enter Current Date and Compute Amplitude of Physical Cycle | MMDD.YYYY | B | Amplitude of Physical Cycle |
| 7 | Compute Amplitude of Emotional Cycle | | R/S | Amplitude of Emotional Cycle |
| 8 | Compute Amplitude of Intellectual Cycle | | R/S | Amplitude of Intellectual Cycle |
| 9 | Compute Average Amplitude | | R/S | Average Amplitude |

Example: Fred was born on May 2, 1944. Calculate his biorhythm for March 1, 1977. (Assume the Biorhythm program is still in program memory.)

| **Press** | **Display** | **Comments** |
|---|---|---|
| **502.1944** A | 0. | Enter birth date |
| **301.1977** B | 0.82 | Amplitude of physical cycle |
| R/S | 1.00 | Amplitude of emotional cycle |
| R/S | 0.76 | Amplitude of intellectual cycle |
| R/S | 0.86 | Average amplitude |

Fred seems to be in pretty good shape for whatever he attempts to do. Use this program to determine where you are in your biorhythmic cycles.

When using library programs as subroutines in your own program, be extra careful as to which data registers you use and which ones are used by the library program. If you both attempt to use the same data registers for different purposes, program results can be erroneous.

## Conditional Transfers (Decision Makers)

Other features that are very useful in problem solving, are instructions that are capable of making decisions in your programs. A family of what are called *conditional transfer instructions* make this decision-making process possible. Each time one of them is encountered in a program, it makes some test and decides whether to transfer or not — strictly dependent upon the outcome of the test.

There are three types of conditional transfer instructions, differentiated by *what* they test.

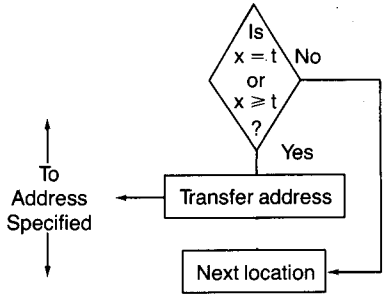
1. Compare display register contents to T-register x=t , x≠t
2. Test contents of data registers 0-9 Dsz
3. Test status of program flags If flg

A transfer address follows each of these instructions. When the answer to the test is "yes" (test positive), transfer is made to that address. If the answer to test is "no" (test negative), the transfer is skipped. For instance, if the test 2nd x=t A is positive, transfer is made to A just as if GTO A had been encountered. If x is not exactly equal the T-register value, no transfer takes place.

### DISPLAY REGISTER VS T-REGISTER

What is this *T-register?* Well, T stands for "TEST" and the register itself is a data memory type of register where numbers can be stored and recalled and compared against the displayed number. The key that gets numbers into and out of the T-register is the "x exchange t" x:t key. This key simply swaps whatever is in the display register, call it x, with t, the contents of the T-register. Initially, 0 is in the T-register. Key 5 into the machine and press x:t . Zero is now in the display and your 5 is in the T-register. Press x:t again and the 5 returns to the display while the zero goes back into the T-register. That's all this key does.

Several instructions are available to compare the current display register value with the contents of the T-register. These "test" instructions are "x equal to t" [x=t] and "x greater than or equal to t" [x≥t]. A program memory address, either absolute or label, must follow each of these instructions whenever they are used. When a test is made, for instance, "Does x = t?" and the answer is "yes," a transfer is made to the address following the test instruction. If not, the address is ignored and processing continues as though no test had even been made. Graphically, here's what happens.

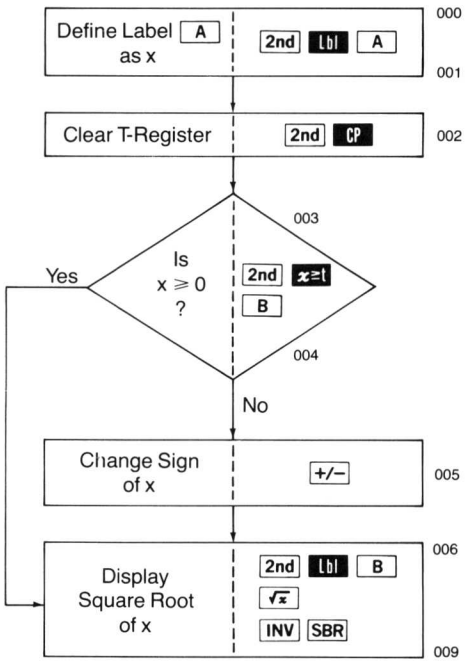



These instructions are designed for use within a program to direct the flow of processing, but they can be used from the keyboard as well. Try these few keystrokes.

| Press | Display | Comments |
|---|---|---|
| 5 [x≠t] | 0. | Put 5 into the T-register |
| 6 [2nd] [x≥t] 123 | 6. | Place 6 in the display and tell the calculator to go to program location 123 if x ≥ t |
| [LRN] | 123 00 | Sure enough transfer was made because 6 is greater than 5 |
| [LRN] 4 [2nd] [x≥t] 111 | 4. | Back to keyboard control and test for x = 4 |
| [LRN] | 123 00 | Still location 123, no transfer was made because 4 is not greater than 5 |
| [LRN] 5 [2nd] [x≥t] 111 | 5. | Back to keyboard control and test for x = 5 |
| [LRN] | 111 00 | Now we've transferred because the display equals the T-register value 5. |

## SQUARE ROOT EXAMPLE

Problem: Find the square root of any number x, entered into the display. If the number is negative, change its sign, then take the square root.



| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 29 | 2nd CP |
| 003 77 | 2nd x≥t |
| 004 12 | B |
| 005 94 | +/− |
| 006 76 | 2nd Lbl |
| 007 12 | B |
| 008 34 | √x |
| 009 92 | INV SBR |

Now you can exercise the program. Enter 4, for example, and press A. 2 is the result. Enter −4 and press A and you get the same thing, 2.

These test instructions can also be used with INV to reverse the conditions of the transfer as shown below.

| Instruction Sequence | Question Asked (Test Made) |
|---|---|
| 2nd x=t | Is the display register value exactly equal to the T-register value? |
| INV 2nd x=t | Is the display register value unequal to the T-register value? |
| 2nd x≥t | Is the display register value greater than or equal to the T-register value? |
| INV 2nd x≥t | Is the display register value less than the T-register value? |

When the answer is "yes" to any of the above questions, the flow of processing transfers to the address that immediately follows the instruction. If the answer is "no," processing simply ignores the accompanying address and goes to the next location of program memory.

# IV

## Programming Considerations

### FLAG OPERATION

What is a program flag and how can it be used in a program? A flag is an internal switch that is either "ON" or "OFF." (Figuratively speaking, a program flag is either raised or lowered.) A flag can be turned on (or set) at some point in a program and tested at a later time. This raising, lowering and testing of flags is independent of the display register and data memory.

Now, when would you want to use a flag? Flags have numerous uses, three 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 led to the present point?

Actually, there are 10 individual flags, numbered 0-9, that you can use. Consequently, with each flag instruction you must specify to which flag you're referring.

The instructions that control flags are defined below.

- To set flag y, press [2nd] [St flg] **y**
- To reset flag y, press [INV] [2nd] [St flg] **y**
- To test flag y and transfer if it is set, press [2nd] [If flg] **y**, then complete the instruction with a transfer address just like the test instructions, mentioned earlier.
- To test flag y and transfer if it is not set, press [INV] [2nd] [If flg] **y**, followed by a transfer address.

These instructions operate from the keyboard as well as in a program. Key in the following and observe the effect of flags.

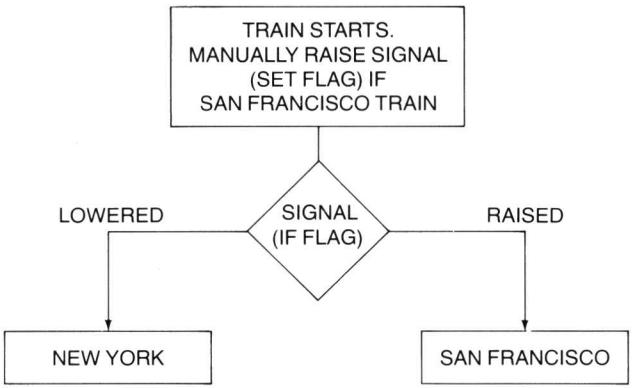
| Press | Display | Comments |
|---|---|---|
| 2nd CP | 0. | Clear program memory and display. This instruction also resets or turns off all program flags and clears the T-register. |
| 2nd St flg 4 | 0. | Set flag number 4 |
| 2nd If flg 4 136 | 0. | Test flag 4, if set, go to location 136 |
| LRN | 136 00 | Transferred to 136 because flag is set |
| LRN 2nd If flg 5 111 | 0. | Test flag 5, if set, go to location 111 |
| LRN | 136 00 | Did not transfer because flag is not set |
| LRN INV 2nd St flg 4 | 0. | Reset flag 4 |
| 2nd If flg 4 222 | 0. | Test flag 4, if set, go to location 222 |
| LRN | 136 00 | Flag is not set, so no transfer is made |
| LRN INV 2nd If flg 4 222 | 0. | Test "If flag is not set — transfer to 222" |
| LRN | 222 00 | Flag not set, so transfer to 222 is performed. |

The Flag test instruction behaves similar to the T-register tests. The difference is that these instructions test flags and the T-register tests compare the display to the T-register. Remember that the transfer address following this instruction can be an absolute address as was used in the exercise above or can be a label of either type (user-defined or common).

Setting a flag that has already been set, resetting a flag that has already been reset, and the testing of a flag have no affect on the status of the flag nor do they affect calculations. All flags can be reset at once with RST or 2nd CP .

Also, note that you cannot directly see if a flag is set (on) or reset (off) in the display (as you can with the T-register or any other data memory). You can only "see" it by testing it.

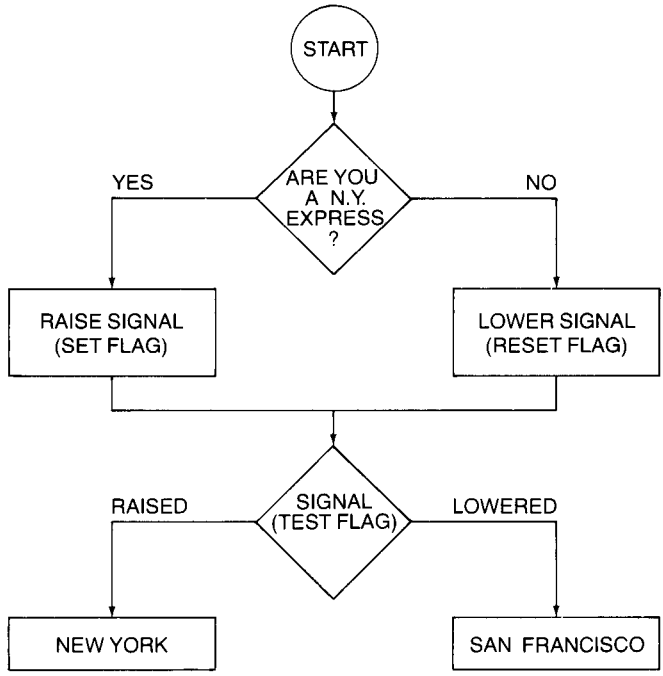
The following situation illustrates the first of the three uses mentioned for flags, how they can be used manually from the keyboard. Suppose you are a train dispatcher in a switch tower. A train is going down the track. It encounters a signal at a junction. If the signal is raised, the train is switched to the San Francisco track. If the signal is lowered, the train is transferred to the New York track. As a dispatcher, you must raise or lower the signal which, in turn, controls which track the train takes. Likewise, you can manually set and reset flags to determine which part of a program is to be executed.




**Manual (Keyboard) Flag Operation**

When working with a program, you can set flags manually from the keyboard to control program operation. For instance, you may have a cost control program in the calculator and a series of credits and debits to be digested by your program. Since debits are to be handled differently than credits, set a flag for the debits and the program should be designed to check the flag and process the incoming entries accordingly.

Now, let's modify the train example to show how the trains themselves could raise and lower flags. This situation demonstrates the principle of program conditions setting flags. Let's say that the New York and San Francisco expresses are to be specially routed to their destinations.




**Automatic (Program) Flag Operation**

The routing system asks each train: "Are you a New York express?" If the answer is "yes," a flag is raised. If the answer is "no," the flag is lowered. This flag is checked later for routing — if it is raised, the train is shunted to New York; if the flag is lowered, it is sent to San Francisco. Similarly, in a program, the most recently calculated value can be asked "Are you negative?" or "Are you greater than 1000?" or many other questions. If the answer is "yes," set a flag and test it later when you need to choose processing options.

The third use of flags gives the program a means of remembering how it reached a given point. This is necessary in situations where what you wish to do depends on which path your program has taken. You recall that the program pointer only knows where it is and has no recollection of how it got there. Such recollective ability is sometimes needed, however, and the program flags provide a convenient way to do this. Just place [2nd] [St flg] **y** in one path and [INV] [2nd] [St flg] **y** in the other path. It is not wise to leave this other path blank as future runs of the program could cause errors if the flag is not reset. Then you can easily determine which path has been followed with [2nd] [If flg] **y**. For instance, a flag can tell you which of two possible interest rates was applied in a program or if a number's sign has to be changed before it can be operated on or lots of other choices.

## SPECIAL FUNCTIONS OF FLAGS

Some flags are internally programmed to perform special functions as follows.

**Flag 7** [2nd] [Op] **18** instructs the calculator to set flag 7 if no error condition exists. Flag 7 is set by the sequence [2nd] [Op] **19** if an error condition does exist (see *SPECIAL CONTROL OPERATIONS on page V-27.)*

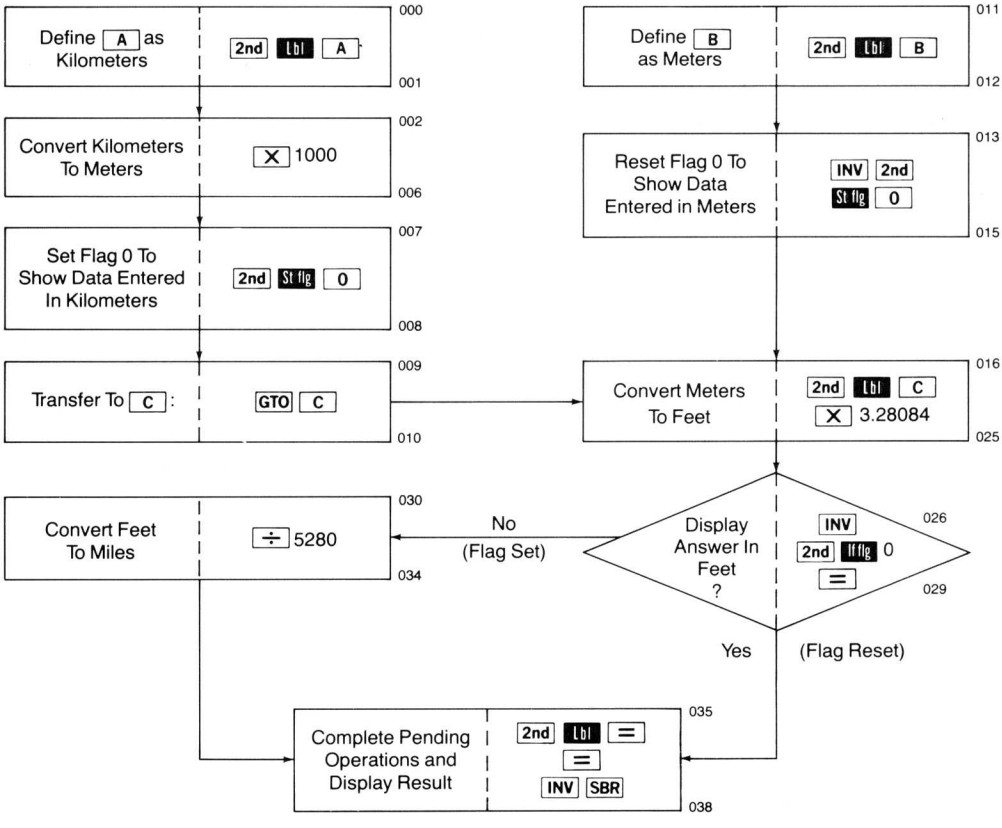
**Flag 8** Setting flag 8 causes the calculator to stop a program if an error occurs while a program is running.

**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 new function or operation. If flag 9 is reset, then results are printed only by a print instruction. Flag 9 may be used normally if you are not using the optional printer.

## METRIC CONVERSION PROGRAM

Create a program that converts meters to feet and kilometers to miles. Now obviously there are quite a few ways to approach this problem. The method used below converts the entered data to feet and then tests to see whether the input data was in kilometers or meters. If the test indicates the entered value was in kilometers, convert to miles; if not, display the answer in feet. $R_1$ is used to store the intermediate result while the test is being made. The conversion factors are:

1 km. = 1,000 meters  1 meter = 3.28084 ft.  1 mile = 5,280 ft.




**Metric Conversion Program**

| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Pointer | | 2nd CP | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Metric Conversion Program | | | |
| 4 | Enter Kilometers<br>OR<br>Enter Meters and Compute Result | Kilometers<br><br>Meters | A<br><br>B | Miles<br><br>Feet |

| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 65 | × |
| 003 01 | 1 |
| 004 00 | 0 |
| 005 00 | 0 |
| 006 00 | 0 |
| 007 86 | 2nd St flg |
| 008 00 | 0 |
| 009 61 | GTO |
| 010 13 | C |
| 011 76 | 2nd Lbl |
| 012 12 | B |
| 013 22 | INV |
| 014 86 | 2nd St flg |
| 015 00 | 0 |
| 016 76 | 2nd Lbl |
| 017 13 | C |
| 018 65 | × |
| 019 03 | 3 |

| Location and Key Code | Key Sequence |
|---|---|
| 020 94 | +/- |
| 021 02 | 2 |
| 022 08 | 8 |
| 023 00 | 0 |
| 024 08 | 8 |
| 025 04 | 4 |
| 026 22 | INV |
| 027 87 | 2nd If flg |
| 028 00 | 0 |
| 029 95 | = |
| 030 55 | ÷ |
| 031 05 | 5 |
| 032 02 | 2 |
| 033 08 | 8 |
| 034 00 | 0 |
| 035 76 | 2nd Lbl |
| 036 95 | = |
| 037 95 | = |
| 038 92 | INV SBR |

**Metric Conversion Program**

Example: Key in the above program and then convert 50 meters to feet and 90 kilometers to miles.

| Enter | Press | Display | Comments |
|---|---|---|---|
| 50 | B | 164.042 | Meters → Feet |
| 90 | A | 55.92340909 | Kilometers → Miles |

DATA REGISTER TRANSFERS — Dsz

This powerful instruction uses the contents of data registers 0-9 to decide whether or not to transfer. Dsz is used primarily for conditional looping so further discussion is postponed until that section.

## Creating Loops

Often in your problem solving, you may require certain processes to be repeated several times in succession to achieve your required result. In this situation you can set up a "looping process." *Looping* is a programming technique where you instruct your calculator to perform a sequence of instructions over and over again until it has done the job you have asked it to do. To create a loop, you simply provide the program with an instruction that resets the program pointer to an earlier location.

### UNCONDITIONAL LOOPING

There are two methods of unconditional looping.

RST loops back to program location 000

GTO loops back to wherever you tell it.

Let's create a program to count by fours. The simple sequence

+ 4 = 2nd Pause RST

should do it if placed at the very start of program memory. After keying this sequence into program memory, exit the learn mode, reset to location 000, enter a starting number and press R/S and watch it count. If you were to place the sequence in program memory starting at location 020, you could replace RST with GTO **020** and accomplish the same thing. Just remember that initially you have to begin execution at location 020.

Be careful with RST because it also resets all flags and clears the subroutine return register.

To exit from a loop, place a transfer inside the unconditional loop to transfer out under the conditions you specify. In the counting by fours example above, let's again count by fours beginning at 0 and stopping at 20.

| Location and Key Code | Key Sequence | Comments |
|---|---|---|
| 000 02 | 2 | |
| 001 00 | 0 | |
| 002 32 | x:t | Store 20 in T-register |
| 003 25 | CLR | Clear display |
| 004 76 | 2nd Lbl | |
| 005 85 | + | Label this segment as + |
| 006 85 | + | |
| 007 04 | 4 | |
| 008 95 | = | |
| 009 66 | 2nd Pause | Display each count |
| 010 67 | 2nd x=t | Test calculated value against T-register |
| 011 00 | 1 | |
| 012 16 | 5 | Skip to location 015 if x = 20 |
| 013 61 | GTO | Otherwise go back to label + |
| 014 85 | + | |
| 015 91 | R/S | Stop when x = t |

Once the program is stored in program memory, just press RST R/S to execute it. Notice that the conditional test in location 010 tests each number that comes through and does nothing until the count reaches 20 then it transfers to 015 and stops. The looping is actually handled by GTO .

## CONDITIONAL LOOPING

The counting example can also be totally controlled by a conditional transfer instruction as was done by [GTO] in the previous example. Again let's count from 0 to 20.

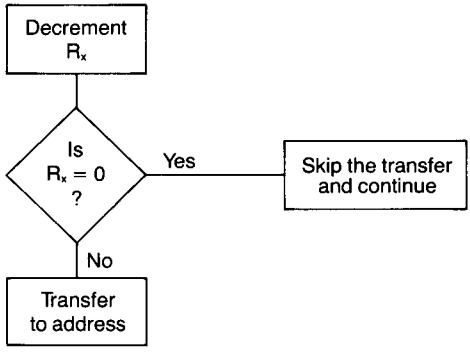
| Location and Key Code | Key Sequence | Comments |
|---|---|---|
| 000 02 | [2] | |
| 001 00 | [0] | |
| 002 32 | [x⇄t] | |
| 003 25 | [CLR] | |
| 004 76 | [2nd] [Lbl] | Label program part as A |
| 005 11 | [A] | |
| 006 85 | [+] | |
| 007 04 | [4] | |
| 008 95 | [=] | |
| 009 66 | [2nd] [Pause] | |
| 010 22 | [INV] | |
| 011 77 | [2nd] [x≥t] | Reverse test and transfer to A if last |
| 012 11 | [A] | calculated value less than 20 |
| 013 91 | [R/S] | Stop when counting reaches 20 |

Here, [INV] [2nd] [x≥t] controls the looping.

## LOOPING WITH THE DSZ CONDITIONAL TRANSFER

Whenever you know how many times a sequence should repeat itself, you can use the "Decrement and Skip on Zero" **DSZ** instruction to handle the looping. The sequence used here is [2nd] [Dsz] X followed by a transfer address. X is the number of one of the 10 data registers 0-9 that can be used with this instruction.

This versatile transfer decreases the magnitude of the contents of data register X by 1 (if the data register contents are less than 1 they are decremented to 0), then tests the contents of register X. (For this discussion, let $R_x$ represent the *contents* of data register X.) If $R_x$ is zero, the transfer address is skipped. Otherwise the address causes the processing sequence to transfer. DSZ decrements register X and skips the transfer on zero. Graphically, this instruction sequence works like this.




Like the other transfer instructions, DSZ can be used from the keyboard as well as in the program. Key in the following and see how.

| Press | Display | Comments |
|---|---|---|
| [2nd] [CP] | 0. | Clear program memory |
| 2 [STO] 6 | 2. | Store 2 in data register 06 |
| [2nd] [Dsz] 6 136 | | Decrements $R_6$ by 1, then asks, "Is $R_6 = 0$?" If no, transfer |
| [LRN] | 136 00 | Transfer made to 136 |
| [LRN] [RCL] 6 | 1. | $R_6$ was 2 and is now 1 because of DSZ |
| [2nd] [Dsz] 6 111 | 1. | Decrement and test again |
| [LRN] | 136 00 | No transfer because $R_6 = 0$ now |
| [LRN] [RCL] 06 | 0. | $R_6$ actually is 0 |

DSZ is actually an effective counter that loops until it counts down to zero, then proceeds to another instruction.

To see how this can be beneficial in a program, let's look at our "counting by" example one more time. We can see that the process of counting by fours to 20 takes 5 passes through the (+4=) loop.

| Location and Key Code | Key Sequence | Comments |
|---|---|---|
| 000 47 | 2nd CMs | Clear all data memories |
| 001 05 | 5 | |
| 002 48 | 2nd Exc | Store 5 in register 00 and clear display |
| 003 00 | 0 | |
| 004 76 | 2nd Lbl | Label this part A |
| 005 11 | A | |
| 006 85 | + | |
| 007 04 | 4 | |
| 008 95 | = | |
| 009 66 | 2nd Pause | Display each count |
| 010 97 | 2nd Dsz | Decrement register 00 and test to see if $R_0$ is zero |
| 011 00 | 0 | |
| 012 11 | A | If $R_0$ is not 0, transfer to A |
| 013 91 | R/S | Stops when $R_0$ is zero |

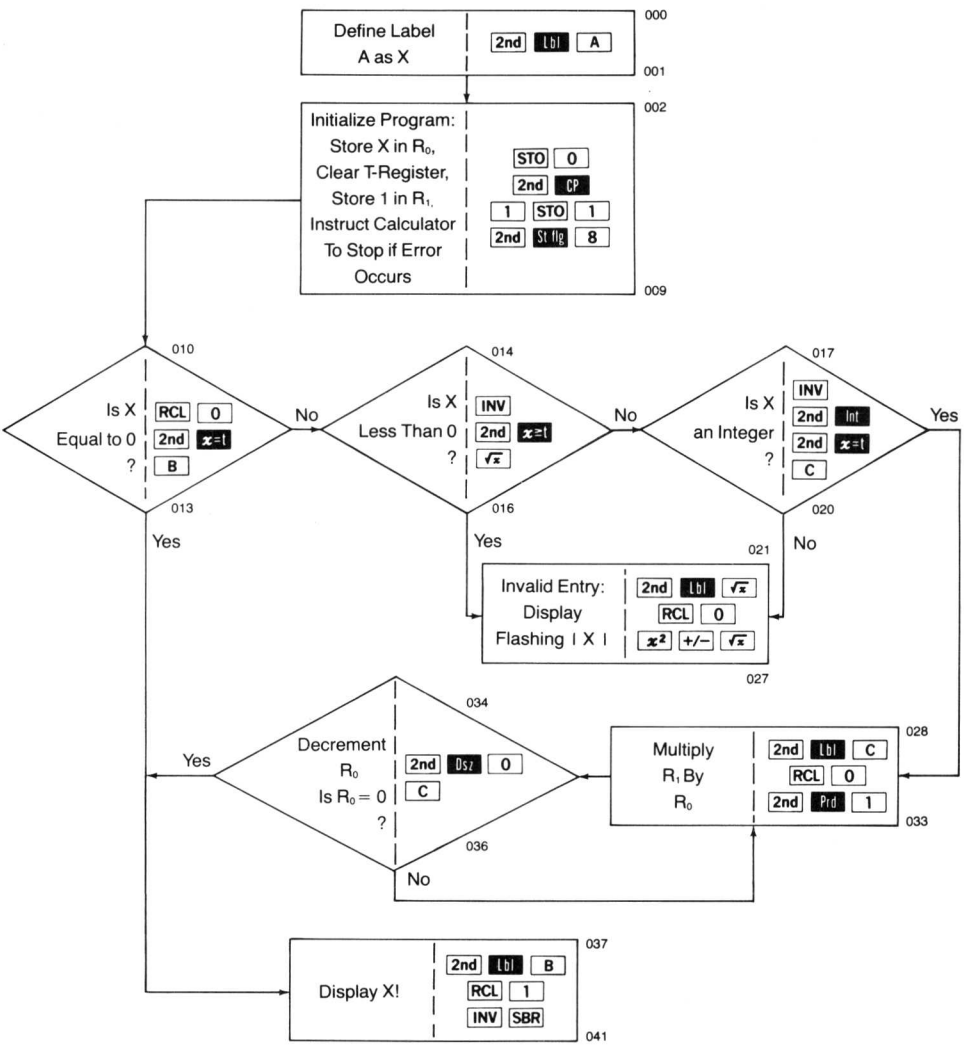
DSZ can *increment* $R_x$ (add 1 to $R_x$) from the negative side of zero as well. A − 5 could have been used just as well in the above example. Also, INV 2nd Dsz still decrements or increments the same as before, but the transfer is skipped on nonzero now instead of zero.

For more details, see *Decrement and Skip on Zero* on page V-63.

This instruction is also valuable when computing a series from 1 to N. You may use DSZ to compute the series by establishing a loop to evaluate the expression for different values of the variable and instructing the calculator to recall the contents of the data register being decremented each time the variable is needed. (Note that the series is actually computed from N to 1 because DSZ decrements.)

X! PROGRAM

Now to exercise the principles of DSZ looping, let's design a program to compute factorials, X!, where $X! = x \cdot (x-1) \cdot (x-2) \cdot \ldots \cdot 2 \cdot 1$. By definition of this function, X must be a positive integer and $0! = 1$.


Define Label A as X
2nd Lbl A
000
001
Initialize Program: Store X in R0, Clear T-Register, Store 1 in R1, Instruct Calculator To Stop if Error Occurs
STO 0
2nd CP
1 STO 1
2nd St flg 8
002
009
010
Is X Equal to 0 ?
RCL 0
2nd x=t
B
013
No
014
Is X Less Than 0 ?
INV
2nd x≥t
√x
016
No
017
Is X an Integer ?
INV
2nd Int
2nd x=t
C
020
Yes
Yes
Yes
No
021
Invalid Entry: Display Flashing I X I
2nd Lbl √x
RCL 0
x² +/- √x
027
034
Decrement R0 Is R0 = 0 ?
2nd Dsz 0
C
036
Yes
No
028
Multiply R1 By R0
2nd Lbl C
RCL 0
2nd Prd 1
033
037
Display X!
2nd Lbl B
RCL 1
INV SBR
041

**X! Program**

| Location and Key Code | Key Sequence | Location and Key Code | Key Sequence |
|---|---|---|---|
| 000 76 | 2nd Lbl | 021 76 | 2nd Lbl |
| 001 11 | A | 022 34 | $\sqrt{x}$ |
| 002 42 | STO | 023 43 | RCL |
| 003 00 | 0 | 024 00 | 0 |
| 004 29 | 2nd CP | 025 33 | $x^2$ |
| 005 01 | 1 | 026 94 | +/− |
| 006 42 | STO | 027 34 | $\sqrt{x}$ |
| 007 01 | 1 | 028 76 | 2nd Lbl |
| 008 86 | 2nd St flg | 029 13 | C |
| 009 08 | 8 | 030 43 | RCL |
| 010 43 | RCL | 031 00 | 0 |
| 011 00 | 0 | 032 49 | 2nd Prd |
| 012 67 | 2nd x=t | 033 01 | 1 |
| 013 12 | B | 034 97 | 2nd Dsz |
| 014 22 | INV | 035 00 | 0 |
| 015 77 | 2nd x≥t | 036 13 | C |
| 016 34 | $\sqrt{x}$ | 037 76 | 2nd Lbl |
| 017 22 | INV | 038 12 | B |
| 018 59 | 2nd Int | 039 43 | RCL |
| 019 67 | 2nd x=t | 040 01 | 1 |
| 020 13 | C | 041 92 | INV SBR |

### X! Program

In the sample program, 1 is stored in $R_1$ so as to allow multiplication by memory arithmetic. As a complete programming exercise, the first three conditional transfers are included to trap out invalid entries. Note that if an invalid entry is made, the error condition created at location 027 halts the program since flag 8 is set earlier in the program. The actual loop occurs between locations 028-036.

Use of this program is very straightforward. Simply enter an x value less than 70 and press [A]. (70! overflows the calculation limits of the calculator.)

Example: Compute 6!; −2!; 0!; 7.3!; 39!

| Enter | Press | Display | Comments |
|---|---|---|---|
| **6** | [A] | 720 | 6! |
| **2** | [+/−] [A] | "2" | Invalid entry |
| | [CLR] | 0 | Clear error |
| **0** | [A] | 1 | 0! |
| **7.3** | [A] | "7.3" | Invalid entry |
| | [CLR] | 0 | Clear error |
| **39** | [A] | 2.0397882 46 | 39! |

NOTE: Quote marks in the display column indicate a flashing display.

## More on Applications

### BOND COST PROGRAM

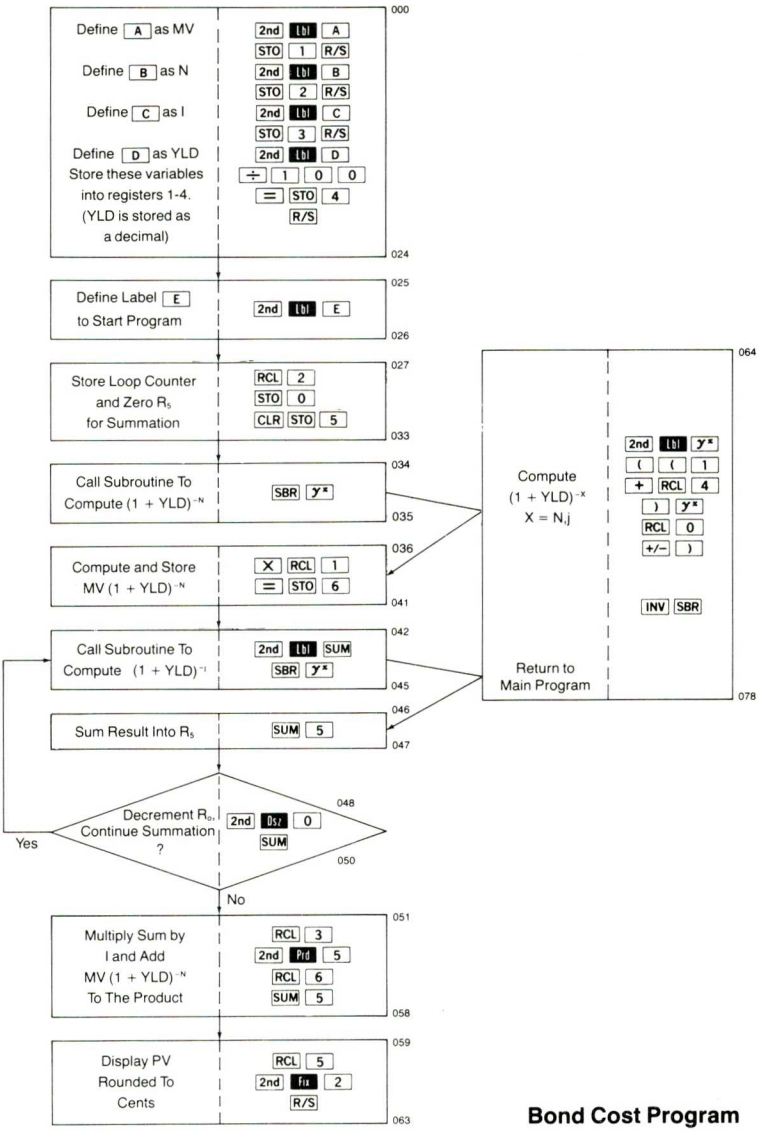
Many investors find buying bonds to be a secure and profitable means of putting their money to work. Others would be interested in buying bonds if they can analyze the potential earnings of their investments. Design a program that may be used to calculate the present value (cost) of a bond with periodic coupons using the formula where the cost of a bond is the sum of the discounted values of the coupons and the maturity value.

$$PV = I \sum_{J=1}^{N} (1 + YLD)^{-J} + MV\,(1 + YLD)^{-N}$$

where:

MV = maturity value
N = number of periods to maturity (j = 1, 2, . . . N)
I = coupon value
YLD = bond yield to maturity (interest per period)
PV = present value or cost of bond

You may write this program using a loop to complete the summation. Since you know the number of loops needed in advance, using the DSZ instruction is the most efficient means of programming the loop, especially since the contents of the data register being decremented may be used to supply the value for j. Also, you may save program space by using a subroutine to evaluate $(1 + YLD)^{-x}$, x = j, N.




**Bond Cost Program**

| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Pointer | | 2nd CP | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Bond Cost Program | | | |
| 4 | Exit Learn Mode | | LRN | 0 |
| 5 | Enter Maturity Value | MV | A | MV |
| 6 | Enter Number of Periods | N | B | N |
| 7 | Enter Coupon Value | I | C | I |
| 8 | Enter Periodic Bond Yield to Maturity | YLD | D | YLD/100 |
| 9 | Compute Present Value | | E | PV |
| | Variables That Do Not Change Need Not Be Reentered For New Problems | | | |

| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 42 | STO |
| 003 01 | 1 |
| 004 91 | R/S |
| 005 76 | 2nd Lbl |
| 006 12 | B |
| 007 42 | STO |
| 008 02 | 2 |
| 009 91 | R/S |
| 010 76 | 2nd Lbl |
| 011 13 | C |
| 012 42 | STO |
| 013 03 | 3 |
| 014 91 | R/S |
| 015 76 | 2nd Lbl |
| 016 14 | D |
| 017 55 | ÷ |
| 018 01 | 1 |
| 019 00 | 0 |
| 020 00 | 0 |
| 021 95 | = |
| 022 42 | STO |
| 023 04 | 4 |
| 024 91 | R/S |
| 025 76 | 2nd Lbl |
| 026 15 | E |
| 027 43 | RCL |
| 028 02 | 2 |
| 029 42 | STO |
| 030 00 | 0 |
| 031 20 | CLR |
| 032 42 | STO |
| 033 05 | 5 |
| 034 71 | SBR |
| 035 45 | $y^x$ |
| 036 65 | × |
| 037 43 | RCL |
| 038 01 | 1 |
| 039 95 | = |
| 040 42 | STO |
| 041 06 | 6 |
| 042 76 | 2nd Lbl |
| 043 44 | SUM |
| 044 71 | SBR |
| 045 45 | $y^x$ |
| 046 44 | SUM |
| 047 05 | 5 |
| 048 97 | 2nd Dsz |
| 049 00 | 0 |
| 050 44 | SUM |
| 051 43 | RCL |
| 052 03 | 3 |
| 053 49 | 2nd P |
| 054 05 | 5 |
| 055 43 | RCL |
| 056 06 | 6 |
| 057 44 | SUM |
| 058 05 | 5 |
| 059 43 | RCL |
| 060 05 | 5 |
| 061 58 | 2nd Fix |
| 062 02 | 2 |
| 063 91 | R/S |
| 064 76 | 2nd Lbl |
| 065 45 | $y^x$ |
| 066 53 | ( |
| 067 53 | ( |
| 068 01 | 1 |
| 069 85 | + |
| 070 43 | RCL |
| 071 04 | 4 |
| 072 54 | ) |
| 073 45 | $y^x$ |
| 074 43 | RCL |
| 075 00 | 0 |
| 076 94 | +/− |
| 077 54 | ) |
| 078 92 | INV SBR |

**Bond Cost Program**

Example: Find the present cost of a bond maturing in 12 years at $20,000 with an annual coupon value of $1,400 and a desired yield of 8%.

| Enter | Press | Display | Comments |
|---|---|---|---|
| **20000** | A | 20000 | MV |
| **12** | B | 12 | N |
| **1400** | C | 1400 | I |
| **8** | D | .08 | YLD |
| | E | 18492.78 | → PV |

A purchase price of $18,492.78 yields 8% annually under these conditions. The total profit of such an investment is 12 × $1,400.00 + ($20,000.00 − $18,492.78) = $18,307.22.

## QUADRATIC EQUATION PROGRAM

A particularly illustrative example of some of the techniques we've been reviewing is the following program designed to handle quadratic equation solutions. It may come in handy also if you find yourself faced with quadratics in problem-solving situations.

Write a program that may be used to calculate the real or complex roots of the equation.

$$ax^2 + bx + c = 0 \; (a \neq 0)$$

The roots $x_1$ and $x_2$ are found by:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

In the event that the value of $b^2 - 4ac$ is positive or equal to zero, the roots are real and are computed according to the above equations. However, if $b^2 - 4ac$ is negative, $x_1$ and $x_2$ are complex roots and must be divided into their real and imaginary parts as demonstrated below.

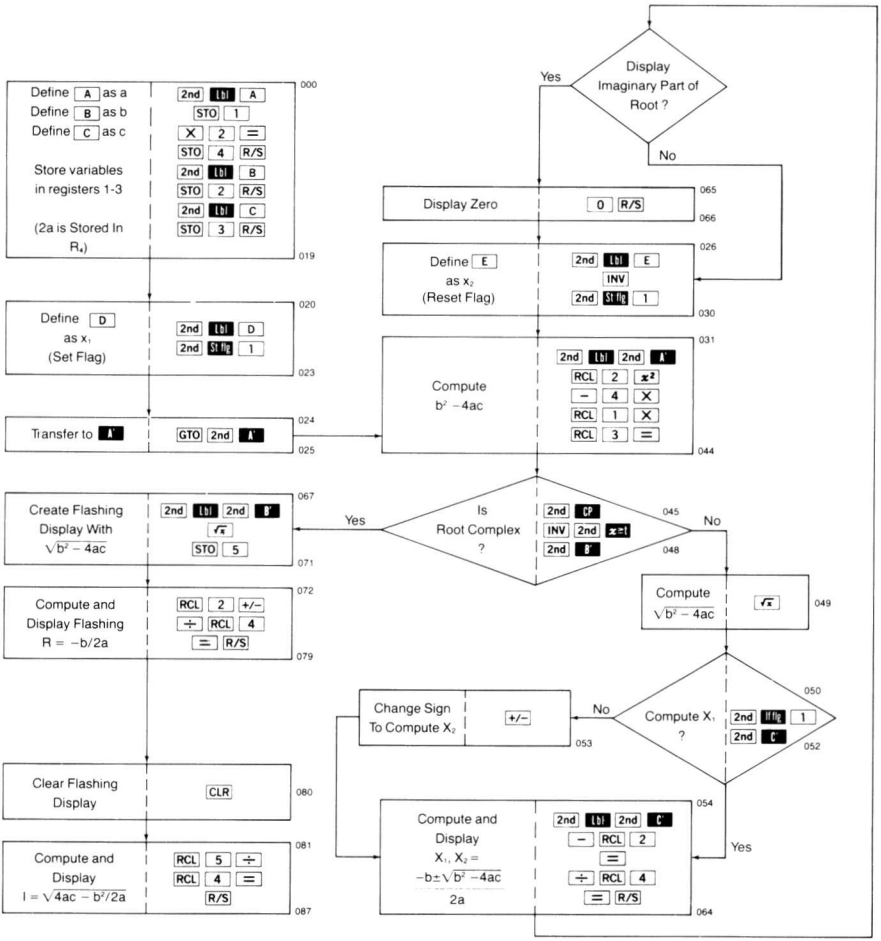
$$x_1 = R + (i \cdot I) \qquad \text{and} \qquad x_2 = R - (i \cdot I)$$

where:

$$R = -b/2a$$
$$i = \sqrt{-1}$$
$$I = \sqrt{4ac - b^2}/2a$$

Since $x_1$ and $x_2$ are calculated using the same basic equations you may save program space by combining the routines and using a flag to indicate which root is being calculated. A separate routine is required to break up complex roots to their real and imaginary parts. You can determine whether the root is complex or real by testing to see if $b^2 - 4ac$ is negative. Note that when the roots are complex you don't need to compute $x_2$ as the values of R and I are the same for both roots.

You should also provide a means of displaying whether a root is real or complex. Since $b^2 - 4ac$ is negative when a root is complex you may create a flashing display by taking the square root of this value before computing the real part of the root. (Note that $\sqrt{4ac - b^2}$ is the actual expression evaluated rather than $\sqrt{b^2 - 4ac}$ when $b^2 - 4ac$ is negative. You may store this result and use it later in determining the imaginary part of the root.) In the sample program below, the imaginary part of the root is determined by pressing [R/S] after computing the real part of the root. As a safeguard, zero is displayed if the root has no imaginary part. This program is not suited for use as a subroutine since [=] and [R/S] have been used.




**Quadratic Equation Program**

| USER INSTRUCTIONS | | | | |
|---|---|---|---|---|
| **Step** | **Procedure** | **Enter** | **Press** | **Display** |
| 1 | Clear Program Memory and Reset Program Pointer | | 2nd CP | |
| 2 | Enter Learn Mode | | LRN | 000 00 |
| 3 | Enter Quadratic Equation Program | | | |
| 4 | Exit Learn Mode | | LRN | 0 |
| 5 | Enter a ($a \neq 0$) | a | A | a |
| 6 | Enter b | b | B | b |
| 7 | Enter c | c | C | c |
| 8 | Compute $x_1$ | | D | $x_1$ (Real) |
| | If Display Flashes Real Part — Root Is Complex — Compute Imaginary Part | | R/S | $x_1$ (Imaginary) |
| 9 | Compute $x_2$ | | E | $x_2$ (Real) |
| | If Display Flashes Real Part — Root Is Complex — Compute Imaginary Part | | R/S | $x_2$ (Imaginary) |



| Location and Key Code | Key Sequence |
|---|---|
| 000 76 | 2nd Lbl |
| 001 11 | A |
| 002 42 | STO |
| 003 01 | 1 |
| 004 65 | × |
| 005 02 | 2 |
| 006 95 | = |
| 007 42 | STO |
| 008 04 | 4 |
| 009 91 | R/S |
| 010 76 | 2nd Lbl |
| 011 12 | B |
| 012 42 | STO |
| 013 02 | 2 |
| 014 91 | R/S |
| 015 76 | 2nd Lbl |
| 016 13 | C |
| 017 42 | STO |
| 018 03 | 3 |
| 019 91 | R/S |
| 020 76 | 2nd Lbl |
| 021 14 | D |
| 022 86 | 2nd St flg |
| 023 01 | 1 |
| 024 61 | GTO |
| 025 16 | 2nd A' |
| 026 76 | 2nd Lbl |
| 027 15 | E |
| 028 22 | INV |
| 029 86 | 2nd St flg |

| Location and Key Code | Key Sequence |
|---|---|
| 030 01 | 1 |
| 031 76 | 2nd Lbl |
| 032 16 | 2nd A' |
| 033 43 | RCL |
| 034 02 | 2 |
| 035 33 | $x^2$ |
| 036 75 | − |
| 037 04 | 4 |
| 038 65 | × |
| 039 43 | RCL |
| 040 01 | 1 |
| 041 65 | × |
| 042 43 | RCL |
| 043 03 | 3 |
| 044 95 | = |
| 045 29 | 2nd CP |
| 046 22 | INV |
| 047 77 | 2nd $x \geq t$ |
| 048 17 | 2nd B' |
| 049 34 | $\sqrt{x}$ |
| 050 87 | 2nd If flg |
| 051 01 | 1 |
| 052 18 | 2nd C' |
| 053 94 | +/− |
| 054 76 | 2nd Lbl |
| 055 18 | 2nd C' |
| 056 75 | − |
| 057 43 | RCL |
| 058 02 | 2 |
| 059 94 | +/− |

| Location and Key Code | Key Sequence |
|---|---|
| 060 95 | = |
| 061 55 | ÷ |
| 062 43 | RCL |
| 063 04 | 4 |
| 064 95 | = |
| 065 91 | R/S |
| 066 00 | 0 |
| 067 91 | R/S |
| 068 76 | 2nd Lbl |
| 069 17 | 2nd B' |
| 070 34 | $\sqrt{x}$ |
| 071 42 | STO |
| 072 05 | 5 |
| 073 43 | RCL |
| 074 02 | 2 |
| 075 94 | +/− |
| 076 55 | ÷ |
| 077 43 | RCL |
| 078 04 | 4 |
| 079 95 | = |
| 080 91 | R/S |
| 081 25 | CLR |
| 082 43 | RCL |
| 083 05 | 5 |
| 084 55 | ÷ |
| 085 43 | RCL |
| 086 04 | 4 |
| 087 95 | = |
| 088 91 | R/S |

**Quadratic Equation Program**

Example: Find the roots of the equation:

$$15x^2 + 3.7x + 2.25 = 0$$

| Enter | Press | Display | Comments |
|---|---|---|---|
| **1.5** | A | 3. | a → 2a |
| **3.7** | B | 3.7 | b |
| **2.25** | C | 2.25 | c |
| | D | − 1.088036702 | Compute $x_1$ (Stable Display Indicates Root Is Real) |
| | E | − 1.378629965 | Compute $x_2$ |

Find the roots of the equation:

$$x^2 + 2x + 17 = 0.$$

| Enter | Press | Display | Comments |
|---|---|---|---|
| **1** | A | 2. | a → 2a |
| **2** | B | 2. | b |
| **17** | C | 17. | c |
| | D | "−1." | Compute Root (Flashing Display Indicates Roots are Complex — R Is Displayed) |
| | R/S | 4. | Compute I |

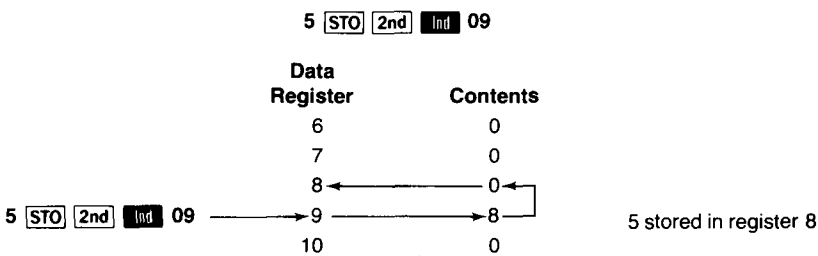
# ADDITIONAL TECHNIQUES

## Programming Indirect Instructions

A whole new set of capabilities can be added to data memory operations, transfer sequences and special control and library program addressing through use of the indirect instruction, [2nd] [Ind]. The basic concept is simple. You go to some data register not to find the information you need, but for *where* to find the information. It's just like telling someone to "Go ask Sam where Fred is" instead of telling the person to "Go and find Fred". You can see that if Sam knows where Fred is, Fred's whereabouts is immediately known to the person asking. But, for someone to just go and find Fred may take hours. In programming, it is sometimes much easier to obtain information indirectly like this. As a matter of fact, for some situations, Fred can never be found directly, so indirect methods are the only means available. Instructions are used indirectly by placing [2nd] [Ind] and a data register number after the instruction. In this data register is found the information needed to complete the instruction.

### DATA REGISTERS ACCESSED INDIRECTLY

All data register instructions (store, recall, exchange, sum, product) can use indirect addressing. Consider the sequence



Let's write a program segment to clear a series of data registers. For simplicity, clear register 1 through X where you can vary X.

| Location and Key Code | Key Sequence | Comments |
|---|---|---|
| 000 76 | 2nd Lbl | To enter X and press A |
| 001 11 | A | |
| 002 42 | STO | Store X in data register 00 |
| 003 00 | 0 | |
| 004 76 | 2nd Lbl | |
| 005 12 | B | |
| 006 25 | CLR | |
| 007 72 | STO 2nd Ind | Zero is to be stored where register 00 says to |
| 008 00 | 0 | |
| 009 97 | 2nd Dsz | DSZ loop on register 00 |
| 010 00 | 0 | |
| 011 12 | B | Go to B if register 00 not zero |
| 012 92 | INV SBR | Halts program when register 00 reaches zero |

First time through the loop, X is in register 00 so the CLR STO 2nd Ind **00** stores a 0 in register X. DSZ then decrements register 00 to (X − 1). Now the indirect store sequence stores its 0 in register (X − 1), etc. The registers have been zeroed in reverse order which really should make little difference. Can you write a program to clear them in numerical order?

Note here the special key code 72 for STO 2nd Ind. Several of the indirect instructions are merged like this to save program space. For a complete list, see *Instruction Codes (Key Codes)* on page V-48.

## INDIRECT TRANSFER STATEMENTS

The usefulness of indirect addressing may be extended to program transfers. Recall that there are two ways to specify a transfer address: by using the absolute location or a label in program memory. Indirect program addressing permits another, more flexible, method. You specify the data register in which the desired absolute address is to be found. A label address cannot be stored in a data register.

Indirect transfer sequences are begun by placing [2nd] [Ind] after either an unconditional transfer statement ( [GTO] , [SBR] ) or a conditional transfer instruction ( [2nd] [x=t] , [2nd] [Dsz] , etc.). The sequence must then be completed with the address of the data register containing the absolute address of the program location you wish to transfer to. Try this sequence from the keyboard.

| Key Sequence | Display | Comments |
|---|---|---|
| **35 [STO] 18** | 35. | Store 35 in data register 18 |
| [GTO] [2nd] [Ind] **18** [LRN] | 035 00 | Program pointer sent to location 035 |

The DSZ instruction may also indirectly specify the register being decremented. That is, sequences such as [2nd] [Dsz] [2nd] [Ind] **12** [2nd] [Ind] **14** are possible when using this instruction. Both the data register used by DSZ and its transfer address can be obtained indirectly.

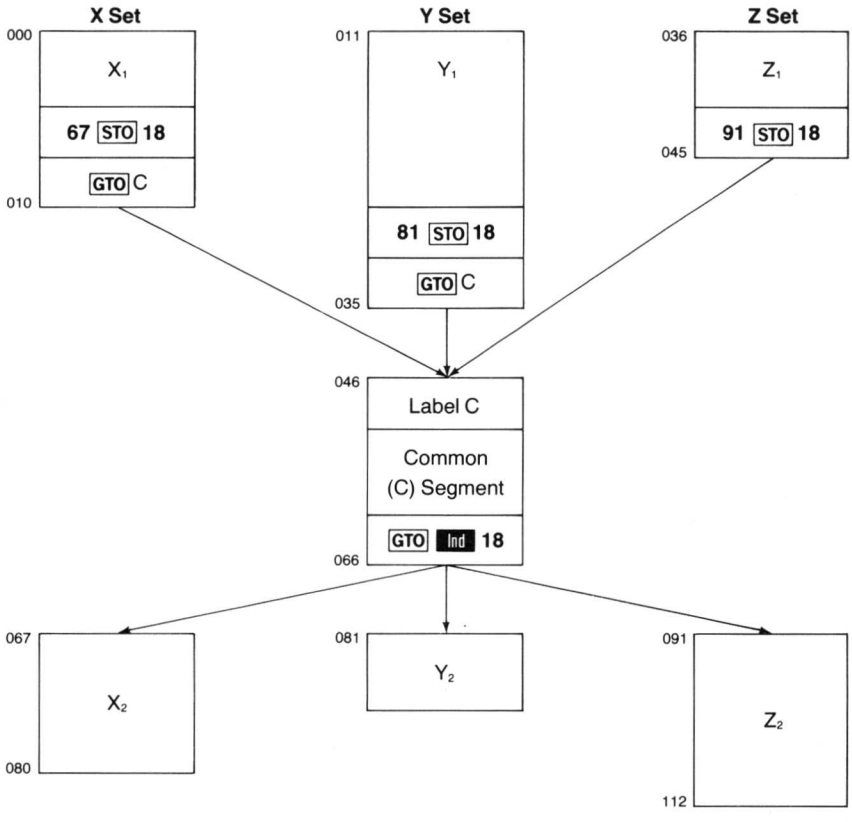
Here is a graphical representation to demonstrate this method of transfer and how it may be used. Assume there are three separate sets of instructions that are to be included in the same program as shown below.

| Set X |
|---|
| $X_1$ |
| C |
| $X_2$ |

| Set Y |
|---|
| $Y_1$ |
| C |
| $Y_2$ |

| Set Z |
|---|
| $Z_1$ |
| C |
| $Z_2$ |

The center portion (C) of each set of instructions is the same, so it would be logical to write the common portion only once. It is an easy matter to use [GTO] instructions at the ends of the segments $X_1$ and $Y_1$ to get to C ($Z_1$ flows directly to C), but how does the program appropriately transfer from C to $X_2$, $Y_2$, and $Z_2$? This problem may be solved using indirect addressing. Simply store the address of the third section before transferring to C and then end C with a [GTO] [2nd] [Ind] instruction. In the diagram, program locations are arbitrarily added to the beginning and end of each segment for illustrative purposes.

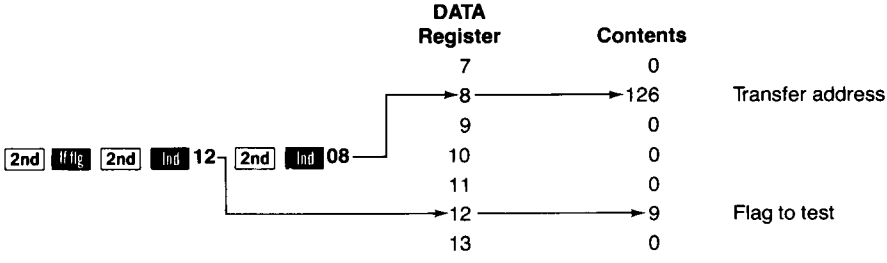



## OTHER FEATURES

You may also use 2nd Ind to indirectly set and reset progam flags and to control the fix-decimal format option of the calculator.

Again, indirect flag control is accomplished by placing the number of the flag in a data register. For example, storing 6 in $R_{12}$ and completing the sequence 2nd St flg 2nd Ind 12 effectively sets flag 6 while 2nd If flg 2nd Ind 12 A branches to label A depending upon the status of flag 6.

Since [2nd] [If flg] is a branching instruction, the branch may also be performed indirectly. Sequences such as [2nd] [If flg] [2nd] [Ind] **12** [2nd] [Ind] **08** are therefore made possible.




An equivalent statement is [2nd] [If flg] **9 126.** Key this example into your calculator to see it work.

| Press | Display | Comments |
|---|---|---|
| **126** [STO] **08** | 126. | Store 126 in register 08 |
| **9** [STO] **12** | 9. | Store 9 in register 12 |
| [2nd] [St flg] **9** | 9. | Set flag 9 |
| [2nd] [If flg] [2nd] [Ind] **12** | | |
| [2nd] [Ind] **8** [LRN] | 126 00 | Transfer is made to location 126 |

Similarly, fix-decimal can be controlled indirectly as the following example shows.

| Press | Display | Comments |
|---|---|---|
| **2** [STO] **12** | 2. | Store 2 in register 12 |
| [2nd] [Fix] [2nd] [Ind] **12** | 2.00 | Calculator placed in fix 2 format |

For more on indirect addressing, see *INDIRECT ADDRESSING* on page V-68.

## Program Optimization

Of the many reasons to optimize a program, two are especially significant. One is to make the program easier to use, and the second is to condense the program to fit in the partition established for program memory.

### PROGRAMMING TECHNIQUES TO SIMPLIFY USAGE

Whether or not a program is easy to use depends upon your own particular needs and preferences. As a general rule, however, a well written program may be easily executed by just a few keystrokes (even by a person other than the programmer).

Many programs require that the entire problem be restarted if a wrong entry or keystroke is made. This can be quite annoying and time consuming, especially when working with long and involved programs. Simplifying error recovery procedures is one way to make a program easier to use. Usually, you may accomplish this by storing and saving the original data. Also, beginning routines that perform memory arithmetic with a [STO] instruction is a good practice as the routine may be rerun without having to clear any data registers.

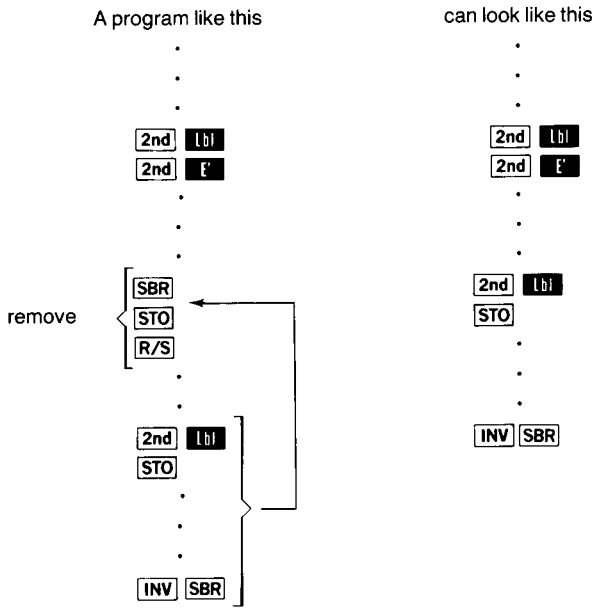
### PROGRAMMING TECHNIQUES FOR MINIMIZING STEPS

Condensing a program to a smaller number of steps is a time-consuming exercise. If a program fits within the program memory partition and operates properly, any time spent to condense the program, in most cases, is unnecessary except for the personal satisfaction of doing it.

When attempting to reduce the number of program steps, you should look for sequences that appear more than once. Then, if these sequences are long enough and needed often enough so that replacing them with subroutines reduces the amount of program space needed, do so.

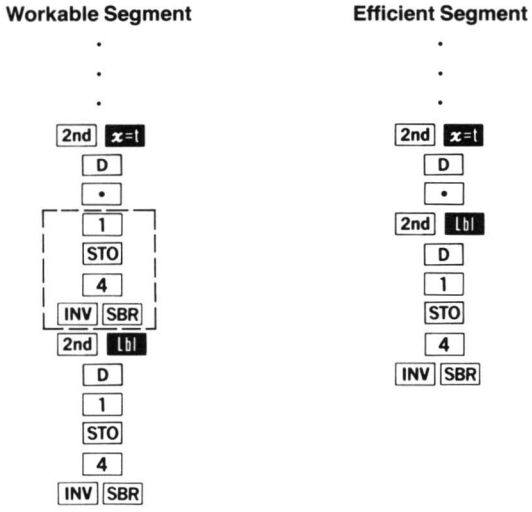
A program requiring numerous subroutines may still exceed the bounds of program memory. Optimization of subroutines thus becomes important.

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



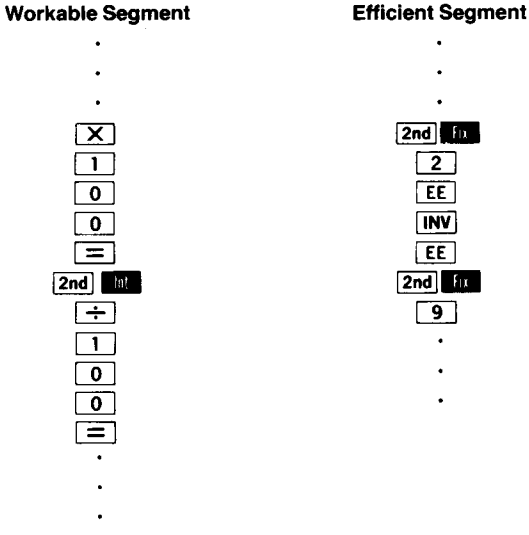

Not only is a savings of several steps realized, but one level of the subroutine return register has been freed. [INV] [SBR] now acts like a [R/S], because the subroutine return register is clear.

As another illustration, consider the two sequences shown below:



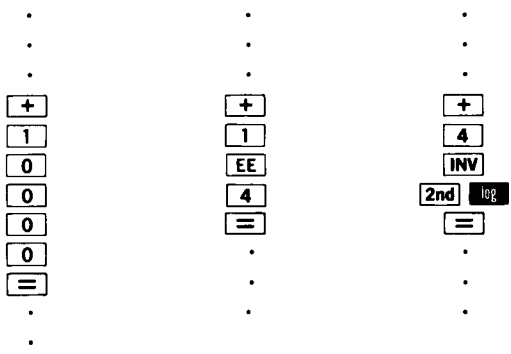

The purpose here is to store a .1 or a 1 depending upon the results of the test. Both of these routines perform the same function; however, the second is four steps shorter than the first as the extraneous instructions enclosed in the box have been eliminated.

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 rounded two-digit value of the number displayed in his calculations. Simply placing the calculator in fix-decimal does not work as most calculations continue to use the full unrounded value.




The purpose and method of the routine on the left is 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 normalizes the display 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.




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 capabilities 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.

If you still have trouble fitting some programs into the allotted space you may be forced to break your program into segments and compute intermediate results before reprogramming the calculator to determine the final solution. Sometimes, however, if you are attempting to program in too straightforward a manner, there is another alternative as illustrated in this next example.

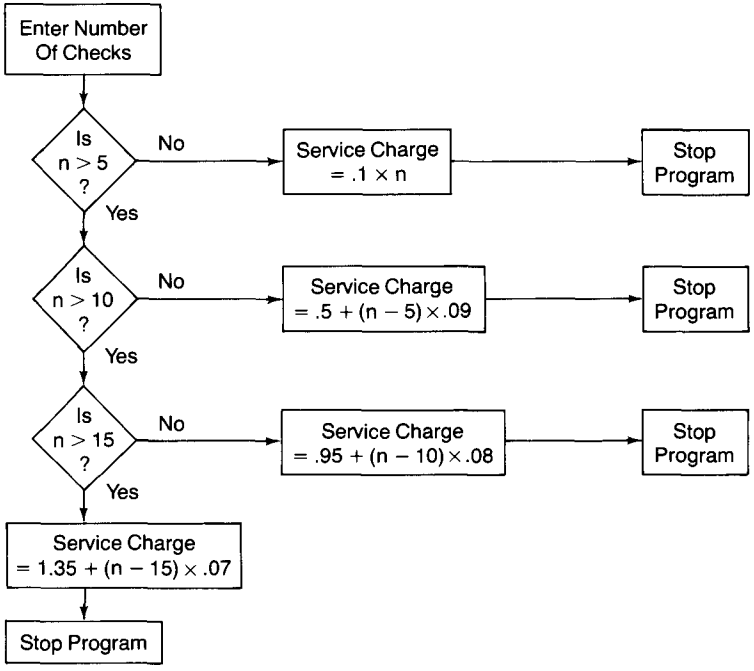
## SERVICE CHARGE PROGRAM

As manager of a prominent local bank, you need a fast and easy method of determining the monthly service charge for the many customers who have accounts with your bank.

The service charge for each account is calculated as follows:

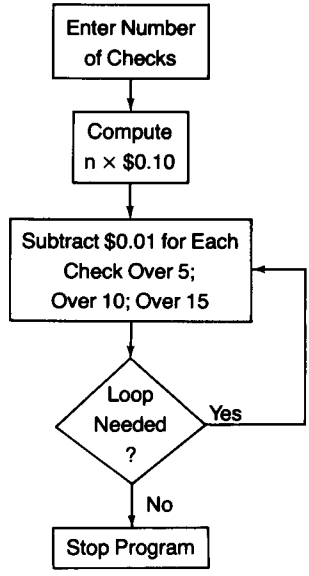
$0.10 per check for the first five checks (1-5),
$0.09 per check for the next five (6-10),
$0.08 per check for the next five (11-15),
$0.07 per check for each check over 15.

A straightforward approach of solving this problem is demonstrated by the following flow diagram.



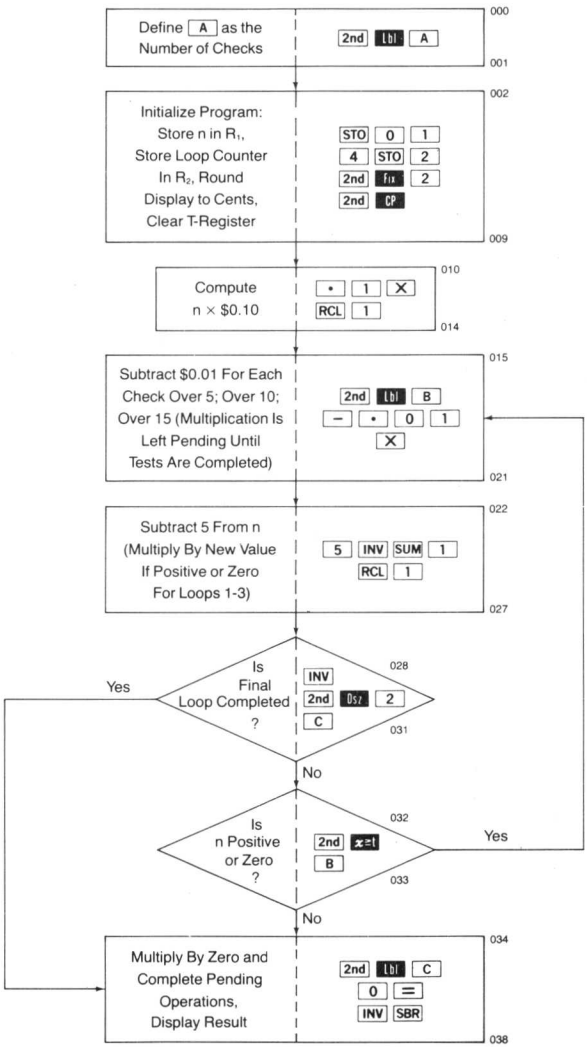

**Service Charge Program** (Basic Approach)

Attempting to write a program following this approach would probably require at least eighty or ninety program locations. Although such a routine could easily fit within program memory, if it were to be used as a subroutine, it may have to be streamlined significantly to allow room for the other program parts. Perhaps another solution would require fewer steps. Consider the following approach.




**Service Charge Program** (Advanced)

At a first glance it would appear that a program following this line of thinking could be easily stored in the program memory of your calculator; however, the reasoning behind some of the sequences used is not readily apparent. Examine the logic here for a moment.




**Service Charge Program**

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 1-3, 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, as the charge on each check over 20 would otherwise 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.

Only two approaches have been made to 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 example demands less time for the program to run. Regardless of the approach you take to programming, bear in mind that the correct method is the one that works best for you.

| Location and Key Code | Key Sequence | Location and Key Code | Key Sequence |
|---|---|---|---|
| 000 76 | 2nd Lbl | 020 01 | 1 |
| 001 11 | A | 021 65 | × |
| 002 42 | STO | 022 05 | 5 |
| 003 01 | 0 1 | 023 22 | INV |
| 004 04 | 4 | 024 44 | SUM |
| 005 42 | STO | 025 01 | 1 |
| 006 02 | 2 | 026 43 | RCL |
| 007 58 | 2nd Fix | 027 01 | 1 |
| 008 02 | 2 | 028 22 | INV |
| 009 29 | 2nd CP | 029 97 | 2nd Dsz |
| 010 93 | . | 030 02 | 2 |
| 011 01 | 1 | 031 13 | C |
| 012 65 | × | 032 77 | 2nd x≥t |
| 013 43 | RCL | 033 12 | B |
| 014 01 | 1 | 034 76 | 2nd Lbl |
| 015 76 | 2nd Lbl | 035 13 | C |
| 016 12 | B | 036 00 | 0 |
| 017 75 | − | 037 95 | = |
| 018 93 | . | 038 92 | INV SBR |
| 019 00 | 0 | | |

**Service Charge Program**

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.

## Programming Techniques for Speed

There are occasions where some time can be saved by reducing 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. Certainly, minimizing the number of transfer statements leads to a faster running program. Therefore, 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 program step may be absolutely addressed with a 3-digit address or with 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 its location. The label search is always begun at location 000 and progresses through program memory until the desired location is found. Then, program execution is continued from that point.

Naturally, when a program is initially entered into the calculator, it is difficult to know ahead of time what the absolute addresses will be. Also, editing a program often causes these addresses to change and significantly increases the difficulty of the task. The best procedure then is to write the original program using labels and convert to absolute addressing only after the program is completely debugged. Again, inserting addresses and deleting labels causes the addresses to change. However, this problem may be overcome using the [2nd] [Nop] instruction.

[2nd] [Nop] simply 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 when program space allows. This technique is illustrated below.

| | | | |
|---|---|---|---|
| | . | | . |
| | . | | . |
| | . | | . |
| 027 | [SBR] | 027 | [SBR] |
| 028 | [In$x$] | 028 | [0] |
| 029 | [2nd] [Nop] | 029 | [7] [5] |
| | . | | . |
| | . | | . |
| | . | | . |
| | . | | . |
| 073 | [2nd] [Lbl] | 073 | [2nd] [Nop] |
| 074 | [In$x$] | 074 | [2nd] [Nop] |
| | . | | . |
| | . | | . |
| | . | | . |
| | . | | . |
| 099 | [GTO] | 099 | [GTO] |
| 100 | [In$x$] | 100 | [0] |
| 101 | [2nd] [Nop] | 101 | [7] [5] |

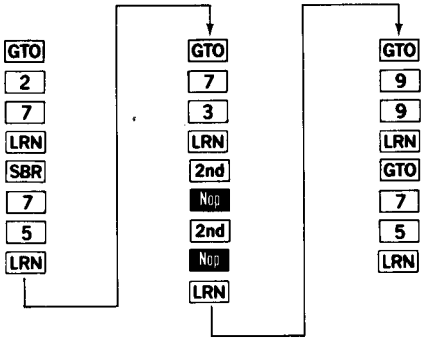
**Label Addressing Converted to Absolute Addressing**

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.

Remember that the absolute address 075 is stored in two program locations rather than three. The first digit, 0, is stored in the first location following the transfer instruction using its standard key code. Then, the second and third digits are merged into the two-digit code "75" and stored in the next program location.

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.



## Codebreaker (Game Program)

This final programming example is not intended for the beginner. The techniques used are designed to make the most efficient use of the array of programming tools available to you. The game is fun to play and there is much to be learned from the program structure itself.

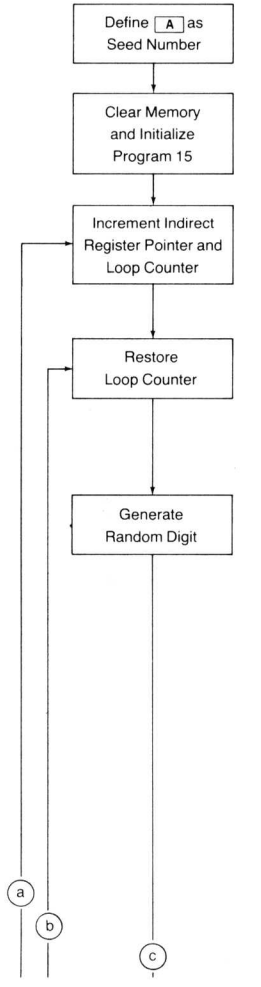
"Codebreaker" is a calculator game where the calculator generates a four-digit secret number and you try to guess it. Zeros are not allowed and no digit may be the same. Even with these restrictions there are 3024 possible codes, making slim your chances of guessing the number on your first guess. Your guess is automatically scored by the calculator. The score is displayed in the form "N". "R" where N is the number of digits in your trial number that appear in the secret number and are positioned correctly, R is the number of digits in your guess which although correct, are improperly placed. For example, if the number generated by the calculator is 8261 and you guess 6285, you receive a score of 1.2. This indicates that one number you guessed is in the right place (the 2) and that two of your other numbers (8 and 6) are present in the secret number, but not in the right place. A score of 4.0 indicates that your guess is correct.

Test your skill by developing a program of your own for this game. Then, study the example given below. Optimize your program to use a minimum number of instructions.

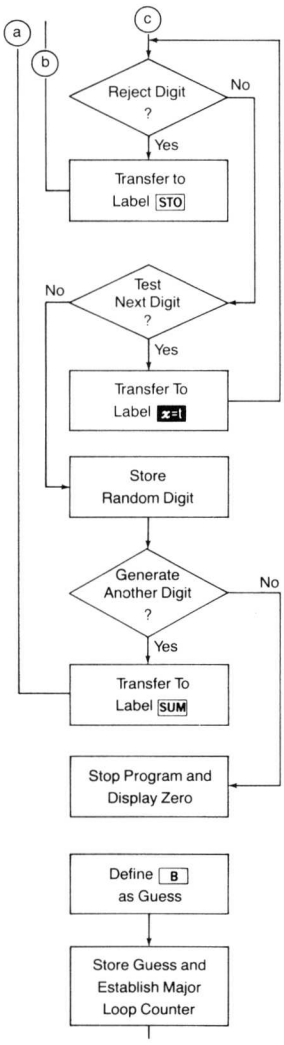
The flow of processing in this example is easy to follow; however, the complexity of the code requires that the flow diagram and its explanation be presented simultaneously.

For the program to derive its secret number, it must have a starting point. A "seed" number is entered at [A] for the calculator to work with.

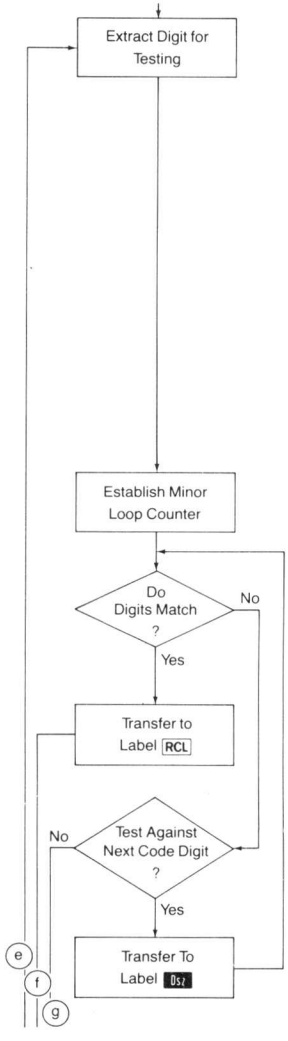




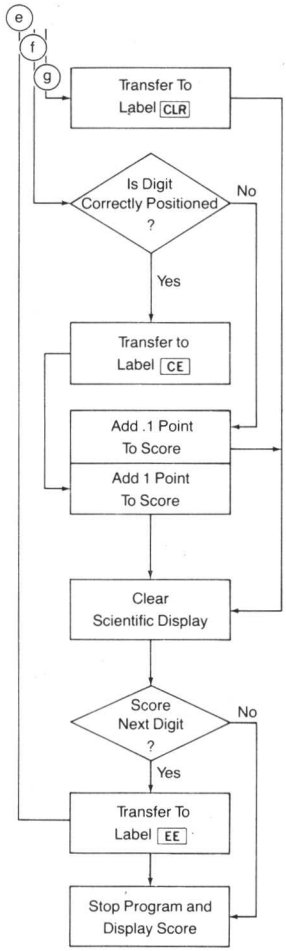

| Comments | Location and Key Code | Key Sequence |
|---|---|---|
| This routine is used to establish the code number. | 000 76<br>001 11 | 2nd Lbl<br>A |
| Program 15 of the Master Library is used to generate random numbers. | 002 47<br>003 36<br>004 15<br>005 15 | 2nd CMs<br>2nd Pgm<br>1 5<br>E |
| The generated digits are indirectly stored according to the contents of $R_5$. Op code 25 is used to increment $R_5$ by 1. | 006 76<br>007 44<br>008 69<br>009 25 | 2nd Lbl<br>SUM<br>2nd Op<br>2 5 |
| A loop is used to test new digits against zero and previously generated digits. To reduce running time, a minimum number of loops is used. | 010 76<br>011 42<br>012 43<br>013 05<br>014 42<br>015 06 | 2nd Lbl<br>STO<br>RCL<br>5<br>STO<br>6 |
| Subroutine D.MS of program 15 is used to generate a random number x where $0 \leq X < 1$. The number is then multiplied by 10 to place it in the proper range and its integer value is placed in the test register. Using routine [C] of program 15 to generate the number would require additional program space to establish the range of the output. Also, routine [C] uses data registers $R_1$-$R_6$ where subroutine D.MS does not. Observe that using the straightforward approach of multiplying by 10 would require an additional program step since the [CLR] instruction is used to remove the scientific display rather than [INV] [EE]. | 016 36<br>017 15<br>018 71<br>019 88<br>020 52<br>021 01<br>022 59<br>023 32<br>024 25 | 2nd Pgm<br>1 5<br>SBR<br>2nd D.MS<br>EE<br>1<br>2nd Int<br>x⇄t<br>CLR |




| Comments | Location and Key Code | Key Sequence |
|---|---|---|
| Since the data registers are initially cleared, the first loop tests the generated digit against zero. The remaining loops test against previously generated digits. If the digit is rejected, a new digit is generated without incrementing $R_5$. $R_6$ serves both as an indirect register pointer and a loop counter. The Dsz instruction also serves a dual purpose in that it controls the looping process and the indirect pointer. | 025 76 | 2nd Lbl |
| | 026 67 | 2nd x=t |
| | 027 73 | RCL 2nd Ind |
| | 028 06 | 6 |
| | 029 67 | 2nd x=t |
| | 030 42 | STO |
| | 031 97 | 2nd Dsz |
| | 032 06 | 6 |
| | 033 67 | 2nd x=t |
| If the digit is accepted, it is indirectly stored using $R_5$ as a pointer. | 034 32 | x:t |
| | 035 72 | STO 2nd Ind |
| | | 5 |
| Once four digits have been accepted, the code is complete. The test is made against 3 rather than 4 for a savings of 1 program step. The alternative is 4 INV 2nd x=t. | 037 43 | RCL |
| | 038 05 | 5 |
| | 039 32 | x:t |
| | 040 03 | 3 |
| | 041 77 | 2nd x=t |
| | 042 44 | SUM |
| | 043 00 | 0 |
| | 044 91 | R/S |
| This routine is used to score the player's guess. | 045 76 | 2nd Lbl |
| | 046 12 | B |
| Two loops are used by this routine. The major loop is run once for each digit in the guess. | 047 42 | STO |
| | 048 12 | 1 2 |
| | 049 04 | 4 |
| | 050 42 | STO |
| | 051 05 | 5 |




| Comments | Location and Key Code | Key Sequence |
|---|---|---|
| This instruction sequence picks off | 052 76 | 2nd Lbl |
| the last digit in the guess and | 053 52 | EE |
| stores it in the T-register so that | 054 43 | RCL |
| it may be compared against the | 055 12 | 1 2 |
| code number. This digit is also | 056 52 | EE |
| removed from the guess so that | 057 01 | 1 |
| the third digit becomes the last | 058 94 | +/− |
| digit for the second loop and | 059 42 | STO |
| so on. The equation used is: | 060 12 | 1 2 |
| | 061 22 | INV |
| Digit = $10 \times$ INV Int $\frac{\text{Guess}}{10}$. | 062 59 | 2nd Int |
| | 063 22 | INV |
| | 064 44 | SUM |
| | 065 12 | 1 2 |
| The EE trick is again used | | |
| for dividing and multiplying | 066 52 | EE |
| by 10. | 067 00 | 0 |
| | 068 00 | 0 |
| | 069 32 | x:t |
| The minor loop is used to test | 070 04 | 4 |
| the above digit against each digit | 071 42 | STO |
| in the code number. | 072 06 | 6 |
| Here the digit selected from the | 073 76 | 2nd Lbl |
| player's guess is compared | 074 97 | 2nd Dsz |
| with the digits appearing in the | 075 73 | RCL 2nd Ind |
| code number. If a match is | 076 06 | 6 |
| found, processing is transferred | 077 67 | 2nd x=t |
| to the scoring section. | | |
| | 078 43 | RCL |
| This instruction makes | 079 97 | 2nd Dsz |
| sure that each digit | 080 06 | 6 |
| guessed is compared | | |
| to each digit of the | | |
| secret number (unless | | |
| a match is found in an | | |
| earlier comparison). | | |
| Again, $R_6$ and DSZ | 081 97 | 2nd Dsz |
| serve dual functions. | | |




| Comments | Location and Key Code | Key Sequence |
|---|---|---|
| If no match occurs, the scoring segment is bypassed. | 082 61 | GTO |
| | 083 25 | CLR |
| The guessed digit is in the right position when the loop counters are at the same level. | 084 76 | 2nd Lbl |
| | 085 43 | RCL |
| | 086 43 | RCL |
| | 087 06 | 6 |
| | 088 32 | x:t |
| | 089 43 | RCL |
| | 090 05 | 5 |
| | 091 67 | 2nd x=t |
| | 092 24 | CE |
| If the guessed digit is in the wrong place score .1 point. If it is correctly positioned, score 1 point, Notice how easily the two routines are combined. | 093 93 | . |
| | 094 76 | 2nd Lbl |
| | 095 24 | CE |
| | 096 01 | 1 |
| | 097 44 | SUM |
| | 098 13 | 1 3 |
| Using CLR in place of INV EE saves 2 steps as it also places 0 in the display register for use below. | 099 76 | 2nd Lbl |
| | 100 25 | CLR |
| | 101 25 | CLR |
| If a major loop has been performed for each digit of the guess the program is terminated and the score is displayed. | 102 97 | 2nd Dsz |
| | 103 05 | 5 |
| | 104 52 | EE |
| In addition to calling the score to the display, the Exc instruction also zeros $R_{13}$ (See location 101) to allow summation when the next guess is entered. Placing the calculator in fix 1 allows R to be displayed even when this value is zero. | 105 48 | 2nd Exc |
| | 106 13 | 1 3 |
| | 107 58 | 2nd Fix |
| | 108 01 | 1 |
| | 109 91 | R/S |

The program designed here requires the entry of a decimal (between 1 and 0) "seed" number at [A]. Once the secret number is derived, a zero appears in the display. Begin guessing by entering your guess and press [B]. The validity of your guess is displayed as previously explained. Let's play a game.

| Enter | Press | Display | Comments |
|---|---|---|---|
| .258 | [A] | 0. | Enter "seed" and wait until secret number determined. |
| 1234 | [B] | 0.1 | First guess |
| 5678 | [B] | 2.1 | Second guess |
| 9238 | [B] | 1.0 | Third guess |
| 5694 | [B] | 1.0 | Fourth guess |
| 5198 | [B] | 2.1 | Fifth guess |
| 5718 | [B] | 4.0 | Sixth guess is correct |

A proficient player seldom needs more than six guesses.

# V THE DETAILS

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

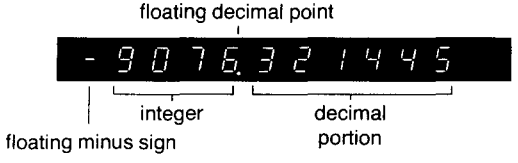
Now for an in-depth analysis of each facet of your calculator. This section is specifically designed as a detailed reference to be used once you have a basic understanding of the calculator's functions. If you do not have a good understanding of the calculator, return to the earlier sections of this manual to obtain this information.

Throughout this section, all discussions about keyboard operations and functions apply both to manual (number by number) calculations as well as to program calculations using those operations.

## BASIC OPERATIONS

### Standard Display

In addition to power-on indication, the display provides numerical information complete with negative sign and decimal point and flashes on and off for an overflow, underflow or error condition. An entry can contain as many as 10 digits. All digits entered after the tenth are ignored.




Any negative number is displayed with a minus sign immediately to the left of the number.


See *Appendix C* for the accuracy of all displayed results.

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## Data Entry Keys

The keys have been selectively positioned on the keyboard to provide for efficient calculator operation. Although many of the operations are obvious, some are not. The following instructions and examples can help you develop skill and confidence in using your calculator.

[0] **THROUGH** [9] **DIGITS** — Enters the numbers 0 through 9.

[•] **— DECIMAL POINT** — Enters the decimal point. The decimal point can be entered wherever needed. If no decimal point is entered, it is assumed to be to the right of the number, and appears when any operation or function key is pressed. A zero precedes the decimal point for numbers less than 1 unless all ten available display digits are used. Trailing zeros on the decimal portion of a number are not normally displayed. Only the first decimal point entered is accepted, all others are ignored. Pressing the decimal point immediately after an exponent entry allows you to alter the mantissa again, like changing its sign.

[2nd] [$\pi$] **— PI** — Enters the value of pi [$\pi$] to 13 significant digits (3.141592653590) for calculations; display indicates the rounded value. [CE] does not remove $\pi$, however, it can be written over by another number.

[+/−] **— CHANGE SIGN** — Instructs the calculator to change the sign of the displayed number. When pressed after [EE], or exponent entry, changes the sign of the exponent.

The procedure for entering a positive number is simply to press the keys in the left to right sequence exactly as the number is written. Each digit entry causes the displayed numbers to shift left as the new digit is entered. Only the first decimal point entered in any single number entry is accepted.

Example: $7.892 - \pi + (-2) = 2.750407346$

| Press | Display |
|---|---|
| **7.892** [−] | 7.892 |
| [2nd] [$\pi$] | 3.141592654 |
| [+] | 4.750407346 |
| **2** [+/−] | −2 |
| [=] | 2.750407346 |

# V

# The Details

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## Clearing Operations

[CE] — **CLEAR ENTRY** — Clears entries made with the digit, decimal point and change-sign keys only when pressed before a function key. This key does not clear calculated results, numbers recalled from memory or $\pi$. [CE] also stops the flashing of the display when needed. Use of this key does not affect pending operations.

[CLR] — **GENERAL CLEAR** — Clears calculations in progress and the display. It resets scientific notation to standard format and stops a flashing display. This key does not affect the contents of the data or program memories, the T-register, angular mode, engineering or fix-decimal display formats or the partition.

The calculator effectively clears itself after most calculations. When the [=] key is pressed to complete a calculation, the answer is displayed and the calculator is ready for the start of a new problem without pressing any of the clear keys. The contents of the data memories are not automatically cleared.

[2nd] [CP] — **CLEAR PROGRAM** — Clears all locations of program memory (and protection), clears the subroutine return register, resets all flags, clears the T-register and resets the program pointer to 000 when pressed from the keyboard. When encountered within a program it only zeros the T-register.

[2nd] [CMs] — **CLEAR DATA MEMORY** — Instructs the calculator to clear all data memory registers as defined by the current partition.

## Dual Function Keys ( [2nd] and [INV] )

Most of your calculator's keys have dual functions. The first function is printed on the key and the second function is written above it. To execute a function shown on a key, simply press the desired key. To use the second function of a key, press the [2nd] key, then press the key immediately below the desired second function. For example, to find the natural logarithm of a number, press [lnx]. To find the common logarithm of a number, press [2nd] [lnx]. In order to distinguish the second function key, in this manual shows it as [2nd] [log]. First function operations, therefore, are indicated by [ ]. Second functions are indicated by [2nd] [ ]. When [2nd] is pressed twice in succession or if a key that does not have a second function is pressed after [2nd], the calculator returns to first function operation.

# The Details



## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

The inverse key [INV] adds additional computing capabilities without increasing the number of keys on the keyboard just like the [2nd] key. When [INV] precedes another key, the purpose of that key is reversed. The inverse can be used with the following keys to obtain the indicated function.

| Function | Inverse Function |
|---|---|
| EE | removes EE |
| ENG | removes ENG |
| Fix | removes Fix |
| log | $10^x$ |
| ln x | $e^x$ |
| $y^x$ | $\sqrt[x]{y}$ |
| Int | fractional part |
| sin | $\sin^{-1}$ |
| cos | $\cos^{-1}$ |
| tan | $\tan^{-1}$ |
| Prod | divide into memory |
| SUM | subtract from memory |
| D.MS | decimal to D.MS |
| P→R | R→P |
| $\Sigma+$ | $\Sigma-$ |
| $\bar{x}$ | standard deviation |
| list | list data registers |
| SBR | return |
| x = t | $x \neq t$ |
| $x \geq t$ | $x < t$ |
| if flg | if no flag |
| st flg | reset flag |
| Dsz | skip on nonzero |
| Write | read |

Programming uses of the [INV] key are discussed in the programming sections where used.

An inverse instruction can be canceled by pressing [INV] a second time if no other key has been pressed. For those keys that do not have an inverse, i.e., [|x|], [LRN], etc., a preceding inverse key is ignored. When used in conjunction with the second function key, the inverse key can be pressed before or after the second function key is pressed, i.e., [INV] [2nd] [log] equals [2nd] [INV] [log]. This is true for the keyboard only. The inverse key must always come first in a program to obtain the desired result. For examples of [INV] uses with a specific key, see the section relating to that key.

# Display Formats

In addition to the versatile 10-digit standard display, there are several other display capabilities that increase the operating range and flexibility of your calculator.

Even though a maximum of 10 digits can be entered or displayed, the internal display register always retains results to 13 digits. The results are then rounded for display only. These extra digits guard the displayed value to insure its accuracy and are not intended to be extended precision. Using these guard digits should be done with extreme care. See *Appendix C* for a detailed discussion of accuracy.

## SCIENTIFIC NOTATION

[EE] — **ENTER EXPONENT** — Instructs the calculator that the subsequent number entry is an exponent of 10. After the [EE] key is pressed, all further results are displayed in scientific notation format until [CLR] is pressed or until the calculator is turned off. Also, [INV] [EE] or [INV] [2nd] [Eng] can remove this format, but only if the displayed number is in the range $\pm 5 \times 10^{-11}$ to $\pm 1 \times 10^{10}$. When [EE] is pressed *after a result* (intermediate or final), internal (guard) digits 11, 12, and 13 are discarded and only the value in the display is used for further calculations.

Any number can be entered as the product of a value (mantissa) and 10 raised to some power (exponent). Just enter the mantissa (up to 8 digits), press [EE], then enter the exponent (any 2 digits).


This capability allows you to work with numbers as small as $\pm 1 \times 10^{-99}$ or as large as $\pm 9.9999999 \times 10^{99}$. Numbers smaller in magnitude than .0000000001 or larger than 9999999999 must be entered in scientific notation. When the results of calculations exceed these limits, the calculator automatically shifts into scientific notation. The entry procedure is to key in the mantissa up to 8 digits (including its sign), then press [EE] and enter the exponent of 10 and its sign.

For example, the number 320,000,000,000 can be written as $3.2 \times 10^{11}$ and can be entered into the calculator as:

| Press | Display |
|---|---|
| [CLR] | 0 |
| **3.2** | 3.2 |
| [EE] | 3.2 00 |
| **11** | 3.2 11 |

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

More than 2 digits can be entered after pressing [EE], but only the last two entered are retained as the exponent. This feature can be used to correct an erroneous exponent entry without having to clear the entire entry using [CE]

In scientific notation, a positive exponent indicates how many places the decimal point of the mantissa should be shifted to the right. If the exponent is negative, the decimal should be moved to the left.

Regardless of how a mantissa is entered for scientific notation, the calculator *normalizes* the number, displaying a single digit to the left of the decimal point, when any function or operation key is pressed.

Example: Enter $6025 \times 10^{20}$

| Press | Display |
|---|---|
| [CLR] | 0 |
| **6025** | 6025 |
| [EE] | 6025 00 |
| **20** | 6025 20 |
| [+] | 6.025 23 |

In scientific notation, the mantissa is limited to 8 digits to allow display space for the exponent. A mantissa resulting from a calculation is also displayed to 8 digits, but internally is carried to 13 digits. This 13-digit value is the one used for all ensuing calculations. See *Appendix C* for more on these extra digits.

**Note: You cannot enter scientific notation format, even though [EE] is pressed, if there are more than 8 mantissa digits entered.** If [EE] is pressed with more than 8 digits in the display, the display goes into scientific notation format when an operation or function key is pressed.

The change sign key can be used to attach a negative sign to the mantissa and to the power-of-ten exponent. Simply press [+/−] after entry of the mantissa to change its sign or after the exponent to change its sign. To change the sign of the mantissa or to enter numbers in its decimal portion after the [EE] key has been pressed, press [•], then enter the mantissa's sign change or additional numbers to the decimal portion.

**Example : Enter $-4.962 \times 10^{-12}$ then correct the exponent and complete the decimal portion of the mantissa to read $-4.96236 \times 10^{12}$.**

| Press | Display | Comments |
|---|---|---|
| [CLR] | 0 | |
| **4.962** [+/−] | −4.962 | Enter mantissa and sign |
| [EE] | −4.962 00 | |
| **12** [+/−] | −4.962−12 | Enter exponent and sign |
| [+/−] | −4.962 12 | Change exponent sign |
| [•] [+/−] | 4.962 12 | Change mantissa sign |
| **36** [+/−] | −4.96236 12 | Complete the mantissa |

# V

## The Details

### AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

Data in scientific notation form can be intermixed with data in standard form. The calculator converts the entered data for proper calculation. After the [EE] key is pressed, the calculator displays all the results in scientific notation format until [CLR], [INV] [EE] or [INV] [2nd] [Eng] is pressed, or until the calculator is turned off. [CE] clears an entry in scientific notation, but the format remains.

Example: $1.816 \times 10^3 - 581.432191 = 1.2345678 \times 10^3 = 1234.567809$

| Press | Display |
|---|---|
| [CLR] | 0 |
| **1.816** [EE] | 1.816 00 |
| **3** [−] | 1.816 03 |
| **581.432191** [=] | 1.2345678 03 |
| [INV] [EE] | 1234.567809 |

When [INV] [EE] is pressed to remove scientific notation and the number is outside of the range $\pm 1 \times 10^{10}$ to $\pm 5 \times 10^{-11}$, the calculator returns to standard format only when or if a calculated result comes into the displayable range.

Example: $(7 \times 10^{11} + 5 \times 10^{10}) \div 25 \div 25 = 1200000000$

| Press | Display |
|---|---|
| **7** [EE] | 7 00 |
| **11** [+] | 7. 11 |
| **5** [EE] | 5 00 |
| **10** [=] [INV] [EE] | 7.5 11 |
| [÷] | 7.5 11 |
| **25** [=] [÷] | 3. 10 |
| **25** [=] | 1200000000. |

If *calculations* exceed 9999999999 or go below .0000000001, the display automatically goes into scientific notation. When this occurs without the [EE] having been pressed during that calculation sequence, the display will automatically revert back to standard display format whenever numerically possible.

To convert a *calculated result* to a scientific notation, there are two approaches. The first is to press [×] 1 [EE] [=] which multiplies the number in the display register by $1 \times 10^0$ and converts the display to scientific notation. The complete 13-digit number is still present. The second method is to press [EE] [=]. You should be careful in using the second method. It has the effect of instructing the calculator to use the ROUNDED quantity being displayed for subsequent calculations discarding the guard digits.

You should avoid using the display commands which use [=] in the middle of a computation. The reason is that the [=] key completes all pending calculations. To avoid this, use these conversion methods only after computations are complete, or else multiply by [×] 1 [EE], followed by another operation.

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

### ENGINEERING NOTATION

This modified form of scientific notation is accessed by pressing [2nd] [Eng]. The displayed value in this format consists of a mantissa and an exponent that are adjusted so that the exponent is a multiple of three ($10^{12}$, $10^{-6}$, etc.) and the mantissa has 1, 2, or 3 digits to the left of the decimal point. This allows the calculator to display results in units that are readily usable such as $10^{-12}$ for picofarads, $10^{-3}$ for millimeters, $10^{6}$ for megohms or $10^{-9}$ for nanoseconds.

Example: What is the diameter of a fibre in micrometer (1 micrometer = $10^{-6}$ meter) whose circumference is $3 \times 10^{-3}$ meters?

$C = \pi d$ $\quad d = C/\pi$

| Press | Display |
|---|---|
| [CLR] [2nd] [Eng] | 0. 00 |
| 3 [EE] | 3 00 |
| 3 [+/−] [÷] | 3.−03 |
| [2nd] [π] [=] | 954.92966−06 |

Pressing [INV] [2nd] [Eng] removes this display format, clearing operations or [INV] [EE] does not affect this format.

### FIX-DECIMAL CONTROL

In standard display format, scientific notation or engineering notation, you can selectively choose the number of digits to display following the decimal point. Pressing [2nd] [Fix], then entering the desired number of decimal places (0 to 8), instructs the calculator to round all displayed results to the selected number of decimal places. This rounding only affects the display, not the display register. So all further calculations use the full unrounded value.

Pressing [2nd] [Fix] 9 or [INV] [2nd] [Fix] returns the calculator to the standard display. Data entries can still be made with 10 digits (8 in scientific notation) with all subsequent calculations using the 13-digit unrounded results, except the DMS-DD conversion that uses the displayed value only. Only the display is altered to the requested number of decimal places unless you press [EE] [INV] [EE] to discard the nondisplayed unit.

Be sure you have removed the display from Engineering Notation format in the previous example.

Example: 2/3 = .6666666667

| Press | Display |
|---|---|
| 2 ÷ | 2. |
| 3 = | .6666666667 |
| 2nd Fix 5 | 0.66667 |
| 2nd Fix 2 | 0.67 |
| 2nd Fix 0 | 1. |
| INV 2nd Fix | .6666666667 |

Remember that the display register value is *rounded* to the desired format.

Example: $1 \times 10^{-3} \div 2 = .0005$

| Press | Display |
|---|---|
| 1 EE | 1 00 |
| 3 +/− ÷ | 1.−03 |
| 2 = | 5.−04 |
| 2nd Fix 2 | 5.00−04 |
| INV EE | 0.00 |
| 2nd Fix 3 | 0.001 |
| 2nd Fix 4 | 0.0005 |
| 2nd Fix 5 | 0.00050 |

Note that the zero that occurs about the middle of the example is not really a zero in the display register. The value just does not round into the fix-2 display format. Always be aware that the display register is not affected by the fix-decimal option.

## FLASHING DISPLAY

The display flashes off and on whenever the limits of the calculator are violated or when an improper mathematical operation is requested. Press CE to stop the flashing without disturbing any calculations in progress. Calculations can continue from this point if the number in the display is still usable. CLR also stops a flashing display, but discards the display value and any calculations in progress. See *Appendix B* for a complete list or error and overflow/underflow conditions and the results they produce.

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## ARITHMETIC CALCULATIONS

This calculator's method of entering numbers and operations allows straightforward entry of most problems just as they are mathematically stated. The calculator remembers each operation and, if necessary, stores it until the standard rules of algebra say that it can be applied.

### Basic Functions — [+] [−] [×] [÷] [=]

To perform simple addition, subtraction, multiplication or division, this calculator with its algebraic operating system allows you to key in the problem just as it is written.

Example: $1.6 \times 10^{-19} \times 6.025 \times 10^{23} = 9.64 \times 10^{4}$

| Press | Display |
|---|---|
| [CLR] | 0 |
| **1.6** [EE] | 1.6 00 |
| **19** [+/−] [×] | 1.6−19 |
| **6.025** [EE] | 6.025 00 |
| **23** [=] | 9.64 04 |

Notice that the [=] key completes the arithmetic operation(s) and displays the final answer.

Pressing [CLR] at the beginning of a new sequence clears any calculations in progress and always ensures that no pending operations remain from prior calculations.

This is not required if the previous problem used [=] to obtain the result. Following [=] with a numeric entry accomplishes the same as pressing [CLR], except that [=] does not remove scientific notation or stop a flashing display.

Pressing any two of the operations keys — [+] [−] [×] [÷] [$y^x$] — in succession causes a flashing display. Also, following any of these with [=] or [)], or preceding with [(] causes a flashing display. See *Appendix B* for more on error conditions.

After a result is obtained in one calculation it may be directly used as the first number in a second calculation. There is no need to reenter the number from the keyboard.

Example: 1.84 + 0.39 = 2.23 then (1.84 + 0.39)/365 = .006109589

| Press | Display | Comments |
|---|---|---|
| **1.84** [+] | 1.84 | |
| **.39** [=] | 2.23 | 1.84 + 0.39 |
| [÷] | 2.23 | |
| **365** [=] | 0.006109589 | 2.23 ÷ 365 |

## Algebraic Operating System Entry Method

Algebraic hierarchy is an essential feature of the Algebraic Operating System method of entering numbers. To efficiently combine operations, the standard rules of algebraic hierarchy have been specifically programmed into the calculator.

These algebraic rules assign priorities to the various mathematical operations. Without a fixed list of priorities, expressions such as $5 \times 4 + 3 \times 2$ could have several meanings:

$$5 \times (4 + 3) \times 2 = 70$$
or $(5 \times 4) + (3 \times 2) = 26$
or $((5 \times 4) + 3) \times 2 = 46$
or $5 \times (4 + (3 \times 2)) = 50$

Algebraic hierarchy rules state that multiplication is to be performed before addition. So algebraically, the correct answer is $(5 \times 4) + (3 \times 2) = 26$. The complete list of priorities for interpreting expressions is:

1. Math Functions
2. Exponentiation ($y^x$) and Roots ($\sqrt[x]{y}$)
3. Multiplication, Division
4. Addition, Subtraction
5. Equals

1. Math functions (trigonometric, logarithmic, square, square root, $e^x$, $10^x$, integer, absolute value, reciprocal and conversions) immediately replace the displayed value with its functional value.
2. Exponentiation ($y^x$) and roots ($\sqrt[x]{y}$) are performed next.
3. Multiplication and division are performed after completing math functions, exponentiation, root extraction and other multiplication and division.
4. Addition and subtraction are performed only after completing all operations through multiplication and division as well as other addition and subtraction.
5. Equals completes all uncompleted operations in the above order.

An operation completes another operation of the same (or lower) priority level. Your calculator knows these rules and applies them to each problem as it is keyed in. Some operations are performed immediately while others are held pending until the rules say to perform them. To illustrate, consider the interpretive order of the following example:

Example: $4 \div 5^2 \times 7 + 3 \times 0.5^{\cos 60^\circ} = 3.241320344$

| Press | Display | Comments |
|---|---|---|
| 4 [÷] | 4. | (4 ÷ ) is stored |
| 5 [$x^2$] | 25. | ($5^2$) special function $x^2$ evaluated immediately |
| [×] | 0.16 | ($4 \div 5^2$) ÷ evaluated because × is same priority as ÷. |
| 7 [+] | 1.12 | × higher priority than + so ($4 \times 5^2 \times 7$) evaluated, 1.12 + stored |
| 3 [×] | 3. | (3×) stored |
| .5 [$y^x$] | 0.5 | .5 $y^x$ stored |
| 60 [2nd] [COS] | 0.5 | Cos 60° evaluated immediately |
| [=] | 3.241320344 | Completes all operations $.5^{\cos 60^\circ}$ evaluated, then $3 \times .5^{\cos 60^\circ}$ next, then this is added to 1.12. |

Thus, by entering the expression as it is written, the calculator correctly interprets it as

$$\{[(4 \div 5^2) \times 7] + (3 \times 0.5^{(\cos 60^\circ)})\}$$

The important things to remember here are that operations are enacted strictly according to their relative priority as stated in the rules. The calculator remembers all stored operations and recalls each and its associated number for execution at exactly the correct time and place. Once familiar with the order of these operations, you will find most problems are extremely easy to solve because of the straightforward manner in which they can be entered into the calculator. Additional control over the order of interpretation is provided through the use of parentheses.

## Parentheses

There are sequences of operations for which *you* may need to instruct the calculator exactly how to evaluate the problem to produce the correct answer. Parentheses give you a way to "cluster" numbers and operations. By placing a series of numbers and operations in parentheses, you are instructing the calculator to interpret this expression first — down to a single number — then proceed.

To illustrate the benefit of parentheses, try the following experiment: Press [(] 5 [×] 7 [)], and you will see the value 35 displayed. The calculator has evaluated 5 × 7 and replaced it with 35 even though the [=] was not pressed. Because of this function of parentheses, the algebraic rules now apply their hierarchy of operations within each set of parentheses. Use of parentheses ensures that your problem can be keyed in just as you would have written it down. The calculator remembers each operation and evaluates each part of the expression as soon as all necessary information is available. When a close parenthesis is encountered, all operations back to the corresponding open parenthesis are completed. You should use parentheses if you have any doubts about how the calculator is going to handle an expression.

Even though expressions are normally written like (3 + 2) (4 + 5) implying a multiply between parentheses sets, you must manually enter a multiply for the operation to take place. *Your calculator does not perform implied multiplication.*

Example: $4 \times (5 + 9) \div (7 - 4)^{(2+3)} = .2304526749$

Key in this expression and follow the path to completion.

| Press | Display | Comments |
|---|---|---|
| 4 [×] [(] | 4. | (4×) stored pending evaluation of parentheses |
| 5 [+] | 5. | (5+) stored |
| 9 [)] | 14. | (5 + 9) evaluated |
| [÷] | 56. | Hierarchy evaluates (4 × 14) |
| [(] | 56. | (56 ÷) stored pending evaluation of parentheses |
| 7 [−] | 7. | (7−) stored |
| 4 [)] | 3. | (7 − 4) evaluated |
| [$y^x$] [(] | 3. | Prepares for exponent |
| 2 [+] | 2. | |
| 3 [)] | 5. | (2 + 3) evaluated |
| [=] | .2304526749 | $(7 - 4)^{(2+3)}$ evaluated then divided into 4 × (5 + 9) |

There are limits on how many operations and associated numbers can be stored. Actually, as many as nine parentheses can be open at any one time and eight operations can be pending, but only in the most complex situations would this limit be approached. If you do attempt to open more than 9 parentheses or if the calculator tries to store more than eight operations, the display flashes.

Example: $5 + \{8/[9 - (2/3)]\} = 5.96$

| Press | Display | Comments |
|---|---|---|
| 5 [+] [(] | 5. | |
| 8 [÷] [(] | 8. | |
| 9 [−] [(] | 9. | |
| 2 [÷] 3 [)] | .6666666667 | (2/3) evaluated |
| [)] | 8.333333333 | [9 − (2/3)] evaluated |
| [)] | 0.96 | 8/[9 − (2/3)] |
| [=] | 5.96 | 5 + {8/[9 − (2/3)]} |

Because the [=] key has the capability to complete all pending operations whenever it is used, it could have been used here instead of the three [)] keys. Try working this problem again and pressing [=] instead of the first [)].

Example: $3 \times \{4^{[2^{(-\sqrt[4]{7})}]}\} = 4.700043401$

| Press | Display | Comments |
|---|---|---|
| [CLR] [(] | 0. | |
| 3 [×] [(] | 3. | |
| 4 [$y^x$] [(] | 4. | |
| 2 [$y^x$] [(] | 2. | |
| 7 [INV] [$y^x$] | 7. | |
| 4 [)] | 1.626576562 | $\sqrt[4]{7}$ |
| [+/−] | −1.626576562 | $-(\sqrt[4]{7})$ |
| [)] | .3238557891 | $2^{-(\sqrt[4]{7})}$ |
| [)] | 1.566681134 | $4^{.323...}$ |
| [)] | 4.700043401 | $3 \times 4^{.323...}$ |

Each time a closed parenthesis is encountered, the contents are evaluated back to the nearest open parenthesis and are replaced with a single value. Knowing this you can structure the order of interpretation for whatever purpose you may want. Specifically, you can check intermediate results.

### DUMMY OPERATION WITH PARENTHESES

An additional technique to use with parentheses is the [CE] *dummy operation.* [CE] has the ability to reenter the display value a second time without having to key in the value again. Specifically, it can bring a value into a parenthesis set if it is needed twice in succession in an expression. Follow the example below.

Example: 3.296214 + (3.296214 × 6) = 23.073498

| Press | Display | Comments |
|---|---|---|
| [CLR] **3.296214** [+] | 3.296214 | |
| [(] [CE] [×] | 3.296214 | [CE] reenters 3.296214 |
| **6** [)] | 19.777284 | |
| [=] | 23.073498 | |

The long value 3.296214 had to be entered only once.

## ALGEBRAIC FUNCTIONS

The simplest operations to describe and understand are single-variable functions. These functions operate on the display register value immediately replacing this value with its corresponding function value. These functions do not interfere with any calculations in progress and can therefore be used at any point in a calculation. The accuracy of these functions is discussed in *Appendix C* and in text where necessary.

### Reciprocal

[1/x] — **RECIPROCAL** — Calculates the reciprocal of the value, x, in the display register by dividing x into 1. A flashing display results if x = 0.

| Press | Display |
|---|---|
| **3.2** [1/x] | 0.3125 |

Note that as soon as one of the math function keys is pressed, the displayed value is immediately replaced with its corresponding function value.

# The Details

V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## Logarithms

[ln x] — **NATURAL LOGARITHM** — Calculates the natural logarithm (base e) of the value, x, in the display register. The display flashes if $x \leq 0$.

[2nd] [log] — **COMMON LOGARITHM** — Calculates the common logarithm (base 10) of the value, x, in the display register. The display flashes if $x \leq 0$.

Example: log (1 + ln 1.7) = .1848697249

| Press | Display |
|---|---|
| [CLR] | 0 |
| [ ( ] 1 [ + ] | 1. |
| 1.7 [ln x] [ ) ] | 1.530628251 |
| [2nd] [log] | .1848697249 |

## Powers of 10 and e

[INV] [ln x] —**NATURAL ANTILOG** ($e^x$) — Calculates the natural antilogarithm $e^x$, of the value, x, in the display register. $-227.9559242 \leq x \leq 230.2585092$ or the display will flash.

[INV] [2nd] [log] —**COMMON ANTILOG** ($10^x$) — Calculates the common antilogarithm $10^x$, of the value, x, in the display register. $-99 \leq x < 99.999999998$ or the display will flash.

Example: $e^{(3 + 10^{0.3})} = 147.7116873$

| Press | Display |
|---|---|
| [CLR] [ ( ] 3 [ + ] | 3. |
| .3 [INV] [2nd] [log] [ ) ] | 4.995262315 |
| [INV] [ln x] | 147.7116873 |

## Angle Calculations

Your calculator provides maximum flexibility when performing calculations involving angles.

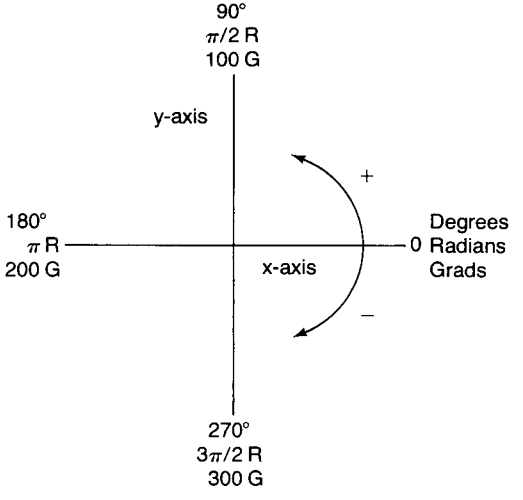
### ANGULAR MODES

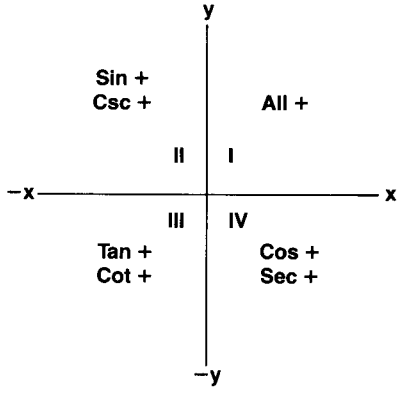
Angles can be measured in decimal degrees, radians or grads (right angle = 90° = $\pi/2$ radians = 100 grads) by pressing either [2nd] [Deg], [2nd] [Rad] or [2nd] [Grad]. The calculator powers-up in the degree mode and stays in that mode until altered by one of the other choices. Once in a certain angular mode, all entered and calculated angles are measured in the units of that mode until another mode is selected or until the calculator is turned off. [CE] and [CLR] do not affect the angular mode.

The angular mode has absolutely no effect on calculations unless the trigonometric functions or polar to rectangular conversions are being performed. Selecting the correct angular mode is easy to do — **AND EASY TO FORGET.** *Neglecting this step is responsible for a large number of errors in operating any calculation device that offers a choice of angular units.*

## TRIGONOMETRIC FUNCTIONS

[2nd] [sin], [2nd] [cos], [2nd] [tan] — **SINE, COSINE, TANGENT** — Calculates the sine, cosine or tangent of the value in the display register. All angles are measured from the X-asis. Measure counterclockwise for positive angles, clockwise for negative, as shown below.




The diagram on the right shows in which quadrant, I-IV, the listed trigonometric functions are positive. Those functions not listed in a particular quadrant have negative values.

When measuring angles, remember that each angle has an equivalent with the opposite sign. For instance $-45° = 315°$.

If your angle is expressed in degrees, minutes and seconds, the [D.MS] key can convert it to decimal form for you. See Conversions on page V-30. Be sure your calculator is in the degree mode. If in doubt, press [2nd] [Deg].

Example: $\sin 30° 13' 48'' + \tan 315° = -0.4965275891$

| Press | Display |
|---|---|
| **30.1348** [2nd] [D.MS] | 30.23 |
| [2nd] [sin] [+] | .5034724109 |
| **315** [2nd] [tan] | −1. |
| [=] | −.4965275891 |

# The Details



## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

Trigonometric values can be calculated for angles greater than one revolution. As long as the trig function is displayed in standard format rather than in scientific notation, all displayed digits are accurate for the range of ± 36,000 degrees, ± 200$\pi$ radians or ± 40,000 grads. In general, the accuracy decreases one digit for each decade outside of this range. If the argument x is greater than ± $3.6 \times 10^{14}$ degrees ($4.0 \times 10^{14}$ grads) or ± $6.2799993 \times 10^{12}$ radians, no partial rotation is recognized.

The other trig functions can be calculated almost as easily. cot = [2nd] [tan] [1/x]
sec = [2nd] [cos] [1/x]
csc = [2nd] [sin] [1/x]

### INVERSE TRIGONOMETRIC FUNCTIONS

[INV] — **INVERSE** — Preceding another key, it reverses the function of that key. When used with the trig functions, the inverse of those functions is obtained. For example, arcsine ($\sin^{-1}$) is obtained by pressing [INV] [2nd] [sin].

The inverse trig functions calculate the angle whose functional value is in the display. The largest angle resulting from an arc function is 180 degrees ($\pi$ radians or 200 grads). Because these functions have many angle equivalents, i.e., arcsin = .5 for 30°, 150°, 390°, etc., the angle returned by each function is restricted as follows:

| ARC FUNCTION | RANGE OF RESULTANT ANGLE |
|---|---|
| arcsin x | 0 to 90°, $\pi/2$ radians, or 100G |
| arcsin (− x) | 0 to −90°, $-\pi/2$ radians, or −100G |
| arccos x | 0 to 90°, $\pi/2$ radians, 100 G |
| arccos (− x) | 90° to 180°, $\pi/2$ to $\pi$ radians, or 100 to 200 G |
| arctan x | 0 to 90°, $\pi/2$ radians, or 100G |
| arctan (− x) | 0 to −90°, $-\pi/2$ radians, or −100G |

For arcsin x and arccos x, $-1 \leq x \leq 1$ or the display will flash.

Example: $\pi/4 + \tan^{-1}(.2\pi) = 1.34638028$

| Press | Display |
|---|---|
| [2nd] [Rad] | 0 |
| [2nd] [π] [÷] | 3.141592654 |
| 4 [+] | .785391634 |
| [(] .2 [×] [2nd] [π] [)] | .6283185307 |
| [INV] [2nd] [tan] | .5609821161 |
| [=] | 1.34638028 |

The selection of the radian mode could have been made at any point before [INV] [2nd] [tan]. It is generally best, though, to select the angular mode at the start of a problem. This assures that the mode is correctly set before you get involved in keying in the problem. The angular mode, whenever selected, only affects angle measurements.

The inverses of the other trig functions can be calculated as follows.

arccot = [1/x] [INV] [2nd] [tan]
arcsec = [1/x] [INV] [2nd] [cos]
arccsc = [1/x] [INV] [2nd] [sin]

## DEGREE, RADIAN, GRAD CONVERSIONS

It is frequently necessary to convert angular values from one unit of measurement to another. Use the following table of conversion factors for the purposes you need.

| FROM \ TO | degrees | radians | grads |
|---|---|---|---|
| degrees | | $\times \frac{\pi}{180}$ | $\div 0.9$ |
| radians | $\times \frac{180}{\pi}$ | | $\times \frac{200}{\pi}$ |
| grads | $\times 0.9$ | $\times \frac{\pi}{200}$ | |

These operations can be performed in any angular mode setting of the calculator.

Example: Convert 120 degrees to radians and grads.

| Press | Display | Comments |
|---|---|---|
| 120 [X] [2nd] [π] [÷] | 376.9911184 | |
| 180 [=] [X] | 2.094395102 | Radians |
| 200 [÷] [2nd] [π] [=] | 133.3333333 | Grads |
| [X] | 133.3333333 | |
| .9 [=] | 120. | Degrees |

Because of the independence of these conversions from the angular mode of the calculator, you must be extremely careful when using the results for further calculations. *The angular mode must be selected to match the units of the results.*

# The Details

V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

### Integer And Absolute Value

[2nd] [Int] — **INTEGER** — Discards the fractional part of the number in the display register. [INV] [2nd] [Int] discards the integer portion of the number in the display register.

[2nd] [|x|] — **ABSOLUTE VALUE** — Makes the value in the display register positive.

Example: Find the absolute value of the integer portion of −13/5.

| Press | Display |
|---|---|
| 13 [+/−] [÷] | −13. |
| 5 [=] | −2.6 |
| [2nd] [Int] | −2. |
| [2nd] [\|x\|] | 2. |

These functions are particularly useful in programming sequences.

Remember that the integer sequence operates on the value in the display register not the displayed value. This means that when [2nd] [Int] is pressed and 4.99999999999 is in the display register (which rounds to 5 in the display) that 4 will be the integer that remains in the display. To obtain a 5 in the display [EE] [INV] [EE] first to truncate the nondisplayed digits **before pressing** [2nd] [Int]

In scientific notation, only the actual fractional part of a number is discarded by the integer key, not the apparent fraction. For instance, $1.2345 \times 10^3$ [2nd] [Int] yields $1.234 \times 10^3$. Actually 1234.5 becomes 1234.

### Square And Square Root

[$x^2$] — **SQUARE** — Calculates the square of the number in the display register. If $|x| \geq 10^{50}$, the display will flash.

[$\sqrt{x}$] — **SQUARE ROOT** — Calculates the square root of the number in the display register. If x is negative, the display will flash.

Example: $[\sqrt{3.1452} - 7 + (3.2)^2]^{1/2} = 2.239078197$

| Press | Display |
|---|---|
| [CLR] [(] | 0. |
| 3.1452 [$\sqrt{x}$] [−] | 1.773471173 |
| 7 [+] | −5.226528827 |
| 3.2 [$x^2$] | 10.24 |
| [)] | 5.013471173 |
| [$\sqrt{x}$] | 2.239078197 |

## Roots and Powers

[$y^x$] — **POWERS** — Raises the display register value, y to the x power. The entry sequence is **y** [$y^x$] **x** followed by an operation or equals key. If $y < 0$, the display will flash.

[INV] [$y^x$] — **ROOTS** — ($\sqrt[x]{y}$ or $y^{1/x}$) — Takes the x root of the value, y, in the display register. The entry sequence is **y** [INV] [$y^x$] **x** followed by an operation or equals key. If $y < 0$ or $x = 0$ or if $y = 0$ and $x < 0$, the display will flash.

These math functions do not act on the display register immediately. They require entry of a second value followed by an operation before the function can be realized.

Example: $\sqrt[3]{2.36^{-.23}} = .9362893421$

| Press | Display | Comments |
|---|---|---|
| **2.36** [$y^x$] | 2.36 | Enter y for $y^x$ |
| **.23** [+/−] | −0.23 | Enter x for $y^x$ |
| [INV] [$y^x$] | .8207865654 | Produces y for $\sqrt[x]{y}$ |
| **3** [=] | .9362893421 | Enter x for $\sqrt[x]{y}$ and produce answer. |

The $y^x$ functions use logarithms to evaluate these functions and the standard mathematical definitions yield the following reactions to various x and y combinations. Quote marks indicate a flashing display.

**Function Reaction**

| y | x | $y^x$ | $\sqrt[x]{y}$ |
|---|---|---|---|
| **0** | **0** | 1. | "1." |
| **0** | **−x** | "9.9999999 99" | "9.9999999 99" |
| **0** | **x** | 0. | 0. |
| **1** | **0** | 1. | "1." |
| **y** | **0** | 1. | "9.9999999 99" |
| **−1** | **0** | "1." | "1." |
| **−y** | **0** | "1." | "9.9999999 99" |
| **−y** | **±x** | "$\lvert y\rvert^{\pm x}$" | "$\sqrt[\pm x]{\lvert y\rvert}$" |

# MEMORY CAPABILITIES

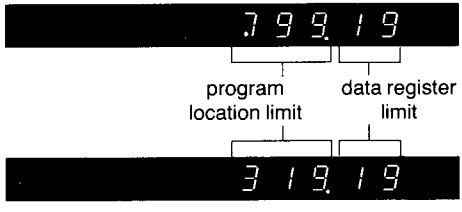
User-accessible data memory registers allow you to store or accumulate data for later use. These storage areas are generally just referred to as data memory or data registers as opposed to program memory where programs are stored. You can use the memory keys at any point in a calculation because they do not affect calculations in progress.

It is usually arbitrary what values are stored where in data memory except that registers 01-06 are used internally if you are working mean and standard deviation calculations. If you want to preserve any values and perform these statistical calculations, use memory registers other than 01-06. If you are using very many data registers for storage, you will probably need some form of bookkeeping to remember what values are stored in which registers.

## Selection of Memory Size (Partitioning)

Throughout this discussion, information about the TI Programmable 58 will follow that for the 59 and are in parentheses in bold type.

When the calculator is first turned on there are 60 (**30**) registers reserved for data storage. The data registers are part of the memory storage area where programs are stored as well as data. You can partition this area 10 registers at a time into different ratios of program to data space according to your needs. Enter the number of sets of 10 registers you need for data storage and press 2nd Op **17.** For example, for 20 data registers, press **2** 2nd Op **17** and the display shows the following:




**TI Programmable 59**

**TI Programmable 58**

This shows that there are 20 registers available for data storage, 00-19, and 800 (**320**) locations set aside for program storage. Remove all fix-decimal, scientific notation and engineering formats before partitioning. Actually, there are 8 program locations in each register not requested for data storage.

To check the current placement of the partition at any time, press 2nd Op **16** and the existing partition is displayed in the format shown above. For more on partitioning, see *Storage Capacity and Partitioning* on page V-42.

Because you can use up to 100 (**60**) data registers, you must specify which data register you are using by entering its 2-digit address, **XX** immediately after pressing any memory related key. Failure to enter a memory address results in a flashing of the current display value. You can, however, use a short form of addressing and enter a single digit address if the address is less than 10 then follow it with a nonnumeric key entry. This totally flexible memory system can manipulate data in a variety of ways.

## Clearing Data Memory

[2nd] [CMs] — **CLEAR DATA MEMORY** — Instructs the calculator to clear all data memory registers as defined by the current partition.

Use of this key does not affect the T-register, program memory, memory partitioning, the display, or calculations in progress.

## Storing And Recalling Data

[STO] **XX** — **STORE** — Stores the display register value into memory register **XX.** Any previously stored data in register is cleared.

[RCL] **XX** — **RECALL** — Recalls and displays the value stored in data register **XX** and retains the value in register **XX.** A recalled number can be used as a number entry in any mathematical expression.

Example: Store and recall 3.012 in memory **22.**

| Press | Display |
|---|---|
| **3.012** [STO] **22** | 3.012 |
| [CLR] | 0 |
| [RCL] **22** | 3.012 |

Use of these keys can save you keystrokes by storing long numbers that are to be used several times.

Example: Evaluate $3x^2 - x - 7.1$ for $x = 2.9467281$

| Press | Display |
|---|---|
| [CLR] **3** [×] | 3. |
| **2.9467281** [STO] **12** | 2.9467281 |
| [$x^2$] [−] | 26.04961949 |
| [RCL] **12** | 2.9467281 |
| [−] **7.1** [=] | 16.00289139 |

The long value of x only had to be entered once. The storage and recall did not interfere with calculations in progress.

You can also use the data memories to hold intermediate results as well as repetitive numbers.

Example: Evaluate $\frac{[\sin(3x/2) - \cos(3x/2)]}{x}$ for x = 20.6821776 degrees

| Press | Display | Comments |
|---|---|---|
| [2nd] [CMs] [(] [(] 3 [X] | 3. | |
| **20.6821776** [STO] **14** | 20.6821776 | Store x in register 14 |
| [÷] 2 [)] [STO] **17** | 31.0232664 | Store 3x/2 in register 17 |
| [2nd] [sin] [−] | .5153861069 | |
| [RCL] **17** | 31.0232664 | Recall 3x/2 from register 17 |
| [2nd] [cos] [)] [÷] | −.3415719789 | |
| [RCL] **14** | 20.6821776 | Recall x from register 14 |
| [=] | −.0165152812 | Answer |

Attempting to use a register beyond the partition causes the display to flash.

## Direct Register Arithmetic

You can store a displayed number at any time during a calculation without affecting the calculation in any way. Additionally, you can add, subtract, multiply and divide the display register value with any data register. The display register itself is not changed. A flashing display following one of these operations indicates that you have exceeded the calculator's operating limit in that register (assuming that you did not call for a register number outside of the current partition that also flashes the display).

[SUM] **XX — MEMORY SUM** — adds the display register value to the contents of data register **XX** and stores the result in **XX.**

[INV] [SUM] **XX — MEMORY SUBTRACT** — Subtracts the display register value from the contents of data register **XX** and stores the result in **XX.**

[2nd] [Prd] **XX — MEMORY PRODUCT** — Multiplies the contents of data register **XX** by the display register value and stores this product in **XX.**

[INV] [2nd] [Prd] **XX — MEMORY DIVIDE** — Divides the contents of data register **XX** by the display register value and stores the result in **XX.**

These capabilities eliminate the lengthy recall perform operations, store-again sequences.

Example: Evaluate $x^2 + 9$ for $x = -1$, 2, and 3 and total the results.

| Press | Display | Memory 3 |
|---|---|---|
| 1 [+/-] [$x^2$] [+] | 1. | 0 |
| 9 [=] [STO] 03 | 10. | 10 |
| 2 [$x^2$] [+] | 4. | 10 |
| 9 [=] [SUM] 03 | 13. | 23 |
| 3 [$x^2$] [+] | 9. | 23 |
| 9 [+] | 18. | 23 |
| [RCL] 3 | 23. | 23 |
| [=] | 41. | 23 |

Notice that the first evaluation was placed in memory 03 using the [STO] key. This is a recommended procedure when performing direct register arithmetic to ensure that you are accumulating only the values you need in that particular register. The [STO] clears any previous content of that register before storing the new value.

Example: The percentage of students completing each year at a particular college is 76.8% first year, 81.3% second year, 92.2% third year and 95.9% last year. What percentage of the students graduate and what percentage complete their third and fourth years?

| Press | Display |
|---|---|
| .768 [X] | 0.768 |
| .813 [X] | 0.624384 |
| .922 [STO] 11 [X] | 0.575682048 |
| .959 [2nd] [Prd] 11 [=] | 0.552079084 |
| [RCL] 11 | 0.884198 |

About 55% of the students that enter the school graduate. Over 88% of those entering their junior year graduate.

## Memory/Display Exchange

[2nd] [Exc] **XX — MEMORY EXCHANGE** — Exchanges the contents of data register **XX** with the display register. The display value is stored and the previously stored value is displayed.

The exchange key has several uses in addition to saving keystrokes. You can use it to examine two results without losing either. Also, numbers can be temporarily stored in **XX** and used as needed.

Example: Evaluate $A^2 + AB + 2B^2$ for A = .258963 and B = 1.255632

| Press | Display | Comments |
|---|---|---|
| **.258963** [STO] **13** | 0.258963 | Store A in register 13 |
| [$x^2$] [+] **1.255632** [X] | 1.255632 | Enter B |
| [2nd] [Exc] **13** | 0.258963 | Store B, recall A |
| [+] **2** [X] | 2. | |
| [RCL] **13** | 1.255632 | Recall B from register 13 |
| [$x^2$] [=] | 3.545447503 | Answer |

## SPECIAL CONTROL OPERATIONS

There are a series of operations accessed through use of the Op key that can greatly boost the capabilities of your calculator. Some of these special operations can be used in any calculator mode while others are designed for a specific mode or for use with optional **PC-100A Print Cradle.**

Each special control operation is called by pressing 2nd Op **nn** where **nn** is the 2-digit code assigned to each operation (short form addressing can be used here). Brief code descriptions are listed below with complete definitions following.

| Code nn | Function |
|---|---|
| **00*** | Initialize print register |
| **01*** | Enters 10 digits in display as 5 alphanumeric codes for far left quarter of print column. |
| **02*** | Enters 10 digits in display as 5 alphanumeric codes for inside left quarter of print column. |
| **03*** | Enters 10 digits in display as 5 alphanumeric codes for inside right quarter of print column. |
| **04*** | Enters 10 digits in display as 5 alphanumeric codes for far right quarter of print column. |
| **05*** | Print the contents of the print register. |
| **06*** | Print last 4 characters of OP **04** with current display value. |
| **07*** | Plot a * in column 0-19 as specified by the display. |
| **08*** | List the labels currently used in program memory. |
| **09** | Bring previously specified library program into program memory. |
| **10** | Apply signum function to display register value. |
| **11** | Calculate variances. |
| **12** | Calculate slope and intercept. |
| **13** | Calculate correlation coefficient. |
| **14** | Calculate new y prime (y′) for an x in the display. |
| **15** | Calculate new x prime (x′) for a y in the display. |
| **16** | Display current partition of memory storage area. |
| **17** | Repartition memory storage area. |
| **18** | If no error condition exists in a program, set flag **7.** |
| **19** | If an error condition exists in a program, set flag **7.** |
| **20-29** | Increment a data register **0-9** by 1. |
| **30-39** | Decrement a data register **0-9** by 1. |

*Designed specifically for use with optional **PC-100A Print Cradle**



## Printer Capabilities — Op 00-08

These control operations are designed for use with the optional PC-100A printer. The printer increases the flexibility of your calculator by providing "hard copy" results of your calculations. The control operations further expand the benefit of the printer by furnishing the options to print alphanumeric messages, to plot data and to list the labels present in a program along with their program locations. The next section on *PRINTER CONTROL* on page VI-7 details the use of each of these special control operations.

## Analysis of Library Program (Downloading) — Op 09

Pressing 2nd Pgm **mm** 2nd Op **09** brings a copy of library program mm into program memory. Here the keyboard operations for program analysis and editing can be used to dissect and alter the program for your particular needs. Beware of the effects that editing has on absolute addressing and combined key codes. When a library program is brought into program memory, it replaces any instructions previously occupying those locations beginning at location 000. There are no provisions for transferring a program from program memory onto the library module. If a program is too large to download into program memory because of either placement of the partition or memory size itself, the library program does not download and the display flashes. The display also flashes if program **mm** is not on the module currently in the calculator or if the module is not in the calculator.

## Signum Function — Op 10

Special control Op **10** applies the signum function to the value "x" currently in the display register and responds with the following.

| Display Register Value x | Display Response |
|---|---|
| $x > 0$ | 1. |
| $x = 0$ | 0. |
| $x < 0$ | −1. |

## Statistics — Op 11-15

The special control operations 11-15 are used for statistical analysis and are discussed fully in the following section on *CONVERSIONS AND STATISTICS.*

## Partitioning — Op 16-17

Pressing 2nd Op **16** immediately displays the current partition location between program and data memories. The last location of program memory and the highest numbered data register currently available are displayed, separated by a decimal point. The TI Programmable 59 is initially partitioned at 479.59, the TI Programmable 58 at 239.29.

To repartition the memory area, enter the number of sets of 10 data registers needed (0-10) and press 2nd Op **17** and the new partition is displayed as above. Remove all fix-decimal, scientific notation and engineering formats before partitioning. See *Storage Capacity and Partitioning* in the GENERAL PROGRAMMING section.

## Test Operations — Op 18-19

Operations 18 and 19 are designed to monitor the error status of a program that is running. When encountered in a program, 2nd Op **18** sets flag **7** if no error condition exists. 2nd Op **19** sets flag **7** if an error condition does exist. Flag **7** can then be monitored by the "if flag" test instruction and appropriate action can then be taken from the results of the test. If the test is false, flag **7** is not changed. See the programming section *Flags* on page V-65 for more information.

## Increment/Decrement Data Registers — Op 20-29/30-39

The OP instruction also allows you to increment or decrement the contents of any data register 0-9 by 1.

To increment register **n** by 1, press 2nd Op **2n**, where **n** is a data register number 0-9.

To decrement register **n** by 1, press 2nd Op **3n**, where **n** is a data register number 0-9.

Each time either of these sequences is pressed or encountered in a program the contents of register **n** are appropriately adjusted by 1 regardless of the content of register **n.**

For example, 2nd Op **34** subtracts 1 from the contents of data register **4**.

# The Details

V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## CONVERSIONS AND STATISTICS

There are several often-used mathematical sequences that have been programmed into your calculator. These operations have been specially designed to provide optimum calculator efficiency by minimizing the number of keystrokes required to execute these iterative sequences.

### Conversions

Your calculator can readily convert between the polar and rectangular coordinate systems. It can also transform angles expressed in degrees, minutes and seconds to decimal degrees and vice versa.

#### ANGLE CONVERSIONS

[2nd] [D.MS] — **DEGREES, MINUTES, SECONDS TO DECIMAL DEGREES** — Converts an angle measured in degrees, minutes and seconds to its decimal degrees equivalent. [INV] [2nd] [D.MS] reverses this conversion. Minutes and seconds can each be any two-digit number. This conversion operates on the displayed value only.

The input format for degrees, minutes and seconds is to a place decimal point between the degrees and minutes, **DD.MMSSsss**. Occasionally, when converting to the degree, minute-second format, minutes or seconds may indicate 60 due to rounding. This simply indicates to round up to the next highest degree (or minute for 60 seconds).

| Press | Display | Comments |
|---|---|---|
| **47.131272** [2nd] [D.MS] | 47.2202 | DD.dddd |
| [INV] [2nd] [D.MS] | 47.131272 | DD. MMSSsss |

**DD** represents degrees and **dd** is for the decimal fraction of a degree. **MM** is minutes and **SSsss** is for seconds and fractional parts of seconds. The trig functions only recognize decimal degrees not minutes and seconds. So, the D.MS conversion must be performed on all angle measurements involving minutes and seconds. Be sure to use two digits each for minutes and seconds. For instance, 5°4′3″ must be entered as 5.0403 to be interpreted correctly.

The D.MS conversion can be used to convert hours, minutes and seconds to a decimal equivalent as well.

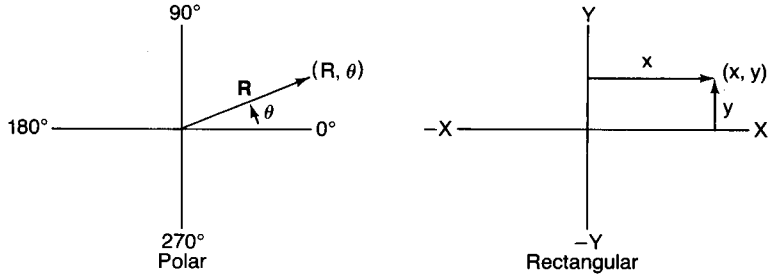
#### POLAR/RECTANGULAR SYSTEM CONVERSIONS

[x⇄t] — **x EXCHANGE t** — Exchanges the display register value **x** with the T-register value **t**. This key is used for data entry with coordinate conversions, statistics and certain conditional testing procedures in programming.

This T-register is independent of all other storage areas, but functions essentially like a data register. It is only externally accessible through the [x⇄t] key that places the display register value into the T-register and brings the contents of the T-register into the display register and the display as well.

[2nd] [P→R] — **POLAR/RECTANGULAR** —Converts polar coordinates to rectangular. [INV] [2nd] [P→R] converts rectangular coordinates to polar. These calculations use 4 pending operation levels internally.

The polar/rectangular conversion is used to convert from polar coordinates which describe any point by a radius "**R**" and an angle theta "$\theta$" to rectangular coordinates which describe any point by two vectors "x" and "y" measured at right angles to each other.




**Polar To Rectangular Sequence**

- Enter "R"
- Press [x⇄t]
- Enter "$\theta$"
- Press [2nd] [P→R] to display "y"
- Press [x⇄t] to display "x"

**Rectangular to Polar Sequence**

- Enter "x"
- Press [x⇄t]
- Enter "y"
- Press [INV] [2nd] [P→R] to display "$\theta$"
- Press [x⇄t] to display "R"

*Be sure to set the desired* **angular mode** *for $\theta$ whether input or calculated.*

The "$\theta$" calculated from the rectangular to polar sequence is:

$$\left.\begin{array}{l}-90° \\ -\pi/2 \text{ rad} \\ -100 \text{ grad}\end{array}\right\} \leq \theta < \left\{\begin{array}{l}270° \\ 3\pi/2 \text{ rad} \\ 300 \text{ grad}\end{array}\right.$$

This says that calculated angles that occur in the fourth quadrant are displayed as negative angles.

This conversion routine monitors the angular mode of the calculator to determine the angular units desired for both entry and retrieval of data.

Example: Convert to rectangular coordinates:
**R** = 5, $\theta$ = 30°

Set angular mode to degrees.

| Press | Display | Comments |
|---|---|---|
| 5 [x⇄t] | 0. | Enter "**R**" |
| 30 [2nd] [P→R] | 2.5 | Enter "$\theta$", display value of "y" |
| [x⇄t] | 4.330127019 | Value of "x" |

Example: Convert to polar coordinates (radians):

$x = 3, y = 4$

| Press | Display | Comments |
|---|---|---|
| [CLR] [2nd] [Rad] | 0 | Set angular mode to radians |
| 3 [x⇄t] | 0. | Store "x" |
| 4 [INV] [2nd] [P→R] | 0.927295218 | Enter "y", display value of "$\theta$" radians |
| [x⇄t] | 5. | Value of "**R**" |

P⟷R and D.MS⟷DD.dd conversions each use one level of subroutine and up to 4 pending operations.

Polar/rectangular conversions are especially useful for vector computations.

## Statistics

Many times it is desirable to express one variable in terms of another even though the variables may not be well defined functions of each other. These variables can be entered as points on a graph with the so-called independent variable carried on the x-axis and the dependent variable plotted on the y-axis. The collection of points can then be analyzed by finding the mean, standard deviation and variance of each of the variables. A straight line can be fitted to the points (linear regression) with the slope and intercept of the line accessible to the user. From here additional data points can be interpolated or extrapolated and a correlation coefficient is available to tell you how closely the line approximates the collection of data points.

### DATA ENTRY

[2nd] [Pgm] 1 [SBR] [CLR] — Initializes calculator for statistics by zeroing data registers 01-06 and the T-register. A library module must, of course, be in the calculator for this sequence to work. Otherwise, you must clear data registers 01-06 and the T-registers manually or in your program.

[x⇄t] — **x EXCHANGE t** — Exchanges the T-register value **t** with the display register value **x.** In statistics, enters the independent (y) variable into the calculator.

[2nd] [Σ+] — **STATISTICS SUM** — Assimulates each variable pair ($x_i$, $y_i$) into data registers 01-06.

[INV] [2nd] [Σ+] removes unwanted data point (pair of values).

To enter a single variable array of data, key in each value and press [2nd] [Σ+]. A faulty entry is removed by reentering the unwanted value (pair) and pressing [INV] [2nd] [Σ+]. After each entry (or removal) the total number of values entered is displayed.

Two dimensional statistical data are entered using the following key sequence for each data point ($x_i$, $y_i$), i = 1, 2, 3 ... N.

$x_i$ [x⇄t] $y_i$ [2nd] [Σ+]

The data point number i is displayed after each point is entered. To remove an unwanted data point, reenter **both the undesired x and y values again,** but press [INV] immediately before [2nd] [Σ+]. The total number of points N input up to there is automatically decremented by 1.

As each data point is keyed in it is assimilated into data memory registers 01-06 as follows.

| MEMORY REGISTER | CONTENTS | |
|---|---|---|
| 01 | $\Sigma y$ | dependent variable |
| 02 | $\Sigma y^2$ | dependent variable |
| 03 | N | |
| 04 | $\Sigma x$ | independent variable |
| 05 | $\Sigma x^2$ | independent variable |
| 06 | $\Sigma xy$ | |

Data that has already been so grouped can be entered directly into these registers and analysis can begin immediately.

The calculator "SUMS" the incoming values into these memory registers. The contents of these registers should be cleared, to prevent an erroneous accumulation of statistical data. The key sequence [2nd] [Pgm] 1 [SBR] [CLR] should be used to zero registers 01-06 and the T-register at the beginning of data entry.

## MEAN, VARIANCE AND STANDARD DEVIATION

After all the data is entered (or at any intermediate point after 2 or more data points have been entered), the mean, standard deviation and variance of each array of data can be calculated.

[2nd] [$\bar{x}$] — Calculates and displays the mean of the dependent (y-array) data. Press [x⇄t] next to display the mean of the independent (x-array) data.

[INV] [2nd] [$\bar{x}$] — Calculates and displays the standard deviation of the dependent (y-array) data. When [x⇄t] is pressed next, the standard deviation of the independent (x-array) data is displayed.

[2nd] [Op] 11 — Calculates and displays the variance of the dependent (y-array) data. When [x⇄t] is pressed next, the variance of the independent (x-array) data is displayed.

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

The following equations are used by the calculator.

$$\text{Mean of x-array} = \bar{x} = \frac{\Sigma x}{N} \qquad \text{Mean of y-array} = \bar{y} = \frac{\Sigma y}{N}$$

where N is the total number of data points entered.

$$\text{Standard Deviation of x-array} = \sigma_x = \left[ \frac{\Sigma x^2 - \frac{(\Sigma x)^2}{N}}{N-1} \right]^{1/2}$$

$$\text{Standard Deviation of y-array} = \sigma_y = \left[ \frac{\Sigma y^2 - \frac{(\Sigma y)^2}{N}}{N-1} \right]^{1/2}$$

$$\text{Variance of x-array} = \sigma_x^2 = \frac{\Sigma x^2}{N} - \bar{x}^2$$

$$\text{Variance of y-array} = \sigma_y^2 = \frac{\Sigma y^2}{N} - \bar{y}^2$$

For your convenience, the option has been provided to select N or N−1 weighting for standard deviation and variance calculations. N weighting results in a maximum likelihood estimator that is generally used to describe populations, while the N−1 is an unbiased estimator customarily used for sampled data.

The variance function uses N weighting and standard deviation uses N−1 weighting. Variance is the square of the standard deviation by definition. So, to find the variance with N − 1 weighting, press [INV] [2nd] [$\bar{x}$] [$x^2$] and [x⇄t] [$x^2$] and standard deviation with N weighting is likewise found by pressing [2nd] [Op] 11 [$\sqrt{x}$] and [x⇄t] [$\sqrt{x}$]. See the table below for these key sequences.

| Function | Weighting | Key Sequences: y-array | Key Sequences: x-array |
|---|---|---|---|
| Mean | | [2nd] [$\bar{x}$] | [x⇄t] |
| Standard Deviation | N | [2nd] [Op] 11 [$\sqrt{x}$] | [x⇄t] [$\sqrt{x}$] |
| Variance | N | [2nd] [Op] 11 | [x⇄t] |
| Standard Deviation | N−1 | [INV] [2nd] [$\bar{x}$] | [x⇄t] |
| Variance | N−1 | [INV] [2nd] [$\bar{x}$] [$x^2$] | [x⇄t] [$x^2$] |

For single variable data, you do not need to use the [x≷t] key as this key is needed only for entering and displaying attributes of independent variables (x-array). Memory registers 01-06 and the T-register are all still used.

Example: Analyze the following test scores: 96, 81, 87, 70, 93, 77

| Press | Display | Comments |
|---|---|---|
| [2nd] [Pgm] 1 [SBR] [CLR] | 0 | Initialize |
| 96 [2nd] [Σ+] | 1. | 1st Entry |
| 81 [2nd] [Σ+] | 2. | 2nd Entry |
| 97 [2nd] [Σ+] | 3. | 3rd Entry (incorrect) |
| 97 [INV] [2nd] [Σ+] | 2. | Remove 3rd Entry |
| 87 [2nd] [Σ+] | 3. | Correct 3rd Entry |
| 70 [2nd] [Σ+] | 4. | 4th Entry |
| 93 [2nd] [Σ+] | 5. | 5th Entry |
| 77 [2nd] [Σ+] | 6. | 6th Entry |
| [INV] [2nd] [$\bar{x}$] | 9.879271228 | Standard Deviation |
| [2nd] [$\bar{x}$] | 84. | Mean |
| [2nd] [Op] 11 | 81.33333333 | Variance |
| [RCL] 01 | 504. | Total of Scores |

Note that the standard deviation can be calculated first even though the mean is used to determine the standard deviation.

Example: A quantity of tubing has been ordered cut into 100 cm long sections to be checked for length accuracy and uniformity that should be 6.0 gm/cm ± 0.01. The test requires that 6 samples be analyzed at a time.

| Sample | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Length (cm) | 101.3 | 103.7 | 98.6 | 99.9 | 97.2 | 100.1 |
| Weight (gm) | 609 | 626 | 586 | 594 | 579 | 605 |

What is the average weight of the samples taken? How accurate is the cutting machine? What is the uniformity of the samples?

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

| Press | Display | Comments |
|---|---|---|
| [2nd] [Pgm] 1 [SBR] [CLR] | 0 | Initialize |
| 101.3 [x⇄t] | 0. | Enter $x_1$ |
| 609 [2nd] [Σ+] | 1. | Enter $y_1$ |
| 103.7 [x⇄t] | 102.3 | Enter $x_2$ |
| 626 [2nd] [Σ+] | 2. | Enter $y_2$ |
| 98.6 [x⇄t] | 104.7 | Enter $x_3$ |
| 586 [2nd] [Σ+] | 3. | Enter $y_3$ |
| 99.9 [x⇄t] | 99.6 | Enter $x_4$ |
| 594 [2nd] [Σ+] | 4. | Enter $y_4$ |
| 97.2 [x⇄t] | 100.9 | Enter $x_5$ |
| 579 [2nd] [Σ+] | 5. | Enter $y_5$ |
| 100.1 [x⇄t] | 98.2 | Enter $x_6$ |
| 605 [2nd] [Σ+] | 6. | Enter $y_6$ |
| [2nd] [$\bar{x}$] | 599.8333333 | Average of y array (weight) |
| [÷] [x⇄t] | 100.1333333 | Average of x array (length) |
| [=] | 5.990346205 | Average uniformity (gm/cm) |
| [INV] [2nd] [$\bar{x}$] | 17.05774509 | Weight deviation |
| [x⇄t] | 2.240238083 | Length deviation |

The average weight of the samples is about 599.8 grams. The machine is cutting the length to about 100.1 centimeters. The uniformity is better than 5.99 grams/centimeter, easily within the acceptable tolerance. In addition, the standard deviation of the weights of the various pieces is about 17 grams with the lengths deviating by about 2¼ centimeters on the average.

### LINEAR REGRESSION

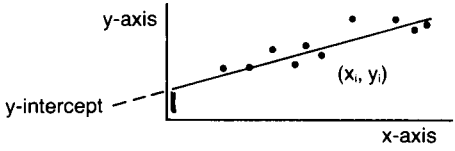
[2nd] [Op] **12** — Calculates and displays the y-intercept of the line fitted to the data points. [x⇄t] when pressed after [2nd] [Op] **12** displays the slope of the line fitted to the data points. Data points that depict a vertical line are a special case that has no y-intercept and has an infinite slope and is an invalid operation for this calculator. Calculating the correlation coefficient ( [2nd] [Op] **13**) detects a vertical or horizontal line by flashing the display.

[2nd] [Op] **13** — Calculates and displays the correlation coefficient of the individual data points in relation to the line fitted to these points. The value will be between −1 and 1 with ±1 being a perfect correlation. If the slope of the line is 0 or infinite, the display flashes. This condition can be monitored in a program through use of [2nd] [Op] **19** and program flags.

[2nd] [Op] **14** — Computes and displays a linear estimate of y′ on the linear regression line corresponding to an x entry from the keyboard. If the previously input data represent a vertical line (infinite slope), new y′ values should not be calculated.

[2nd] [Op] **15** —Computes and displays a linear estimate of x′ on the regression line corresponding to a y entry from the keyboard. If the slope is 0, the display flashes.

Data entry for linear regression is the same as for mean, standard deviation and variance calculations that can also be used here. Actually, once a data set is entered, all the statistical functions can be used to analyze the data. Linear regression allows you to analyze one variable's relationship to another. The method is to perform a least-squares linear regression which is designed to minimize the sum of the squares of the deviations of the actual data points from the straight line of best fit. In practice, a plot of the data points is made and a line is constructed that uniformly divides the data points. The square root of the squares of their perpendicular offsets is minimized.




This line is described by $y = mx + b$, where m is the slope of the line and b is the y-intercept.

Because the data are seldom perfectly linear, you can measure how well the line fitted to the data actually does approximate the data. This measure is called the correlation coefficient and may be calculated from the variables and the linear equation parameters.

The slope and y-intercept of the regression line are determined as follows:

$$\text{slope} = \mathbf{m} = \frac{\Sigma xy - \frac{\Sigma x \, \Sigma y}{N}}{\Sigma x^2 - \frac{(\Sigma x)^2}{N}}$$

$$\text{y-intercept} = \mathbf{b} = \frac{\Sigma y - m\Sigma x}{N}$$

$$\text{The correlation coefficient} = R = \frac{m\,\sigma_x}{\sigma_y}$$

Additional data points can be predicted simply by choosing some new x or y value and the calculator computes a corresponding y or x value on the regression line. This process uses the line equation $y = mx + b$, where m (slope) and b (y-intercept) are determined from the data previously submitted.

# The Details



## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

Example: A life insurance company has found that the volume of sales varies according to the number of sales people employed.

| Number of sales people | 7 | 12 | 3 | 5 | 11 | 8 |
|---|---|---|---|---|---|---|
| Sales in thousands/mo. | 99 | 152 | 81 | 98 | 151 | 112 |

How many sales people does this company need for $200,000 monthly sales? What monthly sales should 15 sales people generate?

| Press | Display | Comments |
|---|---|---|
| [2nd] [Pgm] 1 [SBR] [CLR] | 0 | Initialize |
| 7 [x⇄t] | 0. | First x value |
| 99 [2nd] [Σ+] | 1. | Data point 1 |
| 12 [x⇄t] | 8. | Second x |
| 152 [2nd] [Σ+] | 2. | Data point 2 |
| 3 [x⇄t] | 13. | etc. |
| 81 [2nd] [Σ+] | 3. | |
| 5 [x⇄t] | 4. | |
| 98 [2nd] [Σ+] | 4. | |
| 22 [x⇄t] | 6. | Incorrect entry. |
| 151 [2nd] [Σ+] | 5. | |
| 22 [x⇄t] | 23. | Remove incorrect entry. |
| 151 [INV] [2nd] [Σ+] | 4. | |
| 11 [x⇄t] | 21. | |
| 151 [2nd] [Σ+] | 5. | |
| 8 [x⇄t] | 12. | |
| 112 [2nd] [Σ+] | 6. | |
| 200 [2nd] [Op] 15 | 17.81578947 | People needed for $200,000 sales. |
| 15 [2nd] [Op] 14 | 176.5561798 | Sales for 15 people |
| [2nd] [Op] 12 | 51.66853933 | Y-intercept of line |
| [x⇄t] | 8.325842697 | Slope of line |

The slope and y-intercept have been calculated so that the line can be plotted, if desired. The slope is incremental sales per person. The y-intercept is independent sales.

Of special interest is that by performing any of the math functions on one or both elements of the random-variable pair, other types of regression are available. For example, by taking the logarithm of one of the variables before entering it as a data point, you can obtain a semi-logarithmic curve fit. These variations can be achieved by using natural logarithms, exponentials, roots and powers and reciprocal.

When initially analyzing your data, you must select the type of curve that characterizes your particular situation. Actually you can try several types of curves to see which best fits your needs.

Example: A city published the following census data. Predict the population in the year 1980 and predict the year the population will be 50,000 inhabitants.

| Year | 1930 | 1940 | 1950 | 1960 | 1970 |
|---|---|---|---|---|---|
| Population | 3221 | 5361 | 9212 | 15410 | 27612 |

Population data characteristically follows an exponential curve of the form $y = ae^{bx}$. Taking the log of both sides of this equation yields $\ln y = \ln a + bx$. Therefore, by plotting x vs ln y (semilog), we can plot a straight line.

| Press | Display |
|---|---|
| 2nd Pgm 1 SBR CLR | 0. |
| 1930 x⇄t | 0. |
| 3221 lnx 2nd Σ+ | 1. |
| 1940 x⇄t | 1931. |
| 5361 lnx 2nd Σ+ | 2. |
| 1950 x⇄t | 1941. |
| 9212 lnx 2nd Σ+ | 3. |
| 1960 x⇄t | 1951. |
| 15410 lnx 2nd Σ+ | 4. |
| 1970 x⇄t | 1961. |
| 27612 lnx 2nd Σ+ | 5. |
| 1980 2nd Op 14 INV lnx | 46081.80979 |
| 50000 lnx 2nd Op 15 | 1981.524472 |

The population in 1980 should be about 46,080 and the town should have 50,000 residents by 1981.

## TREND-LINE ANALYSIS

For data that has been collected at periodic intervals, such as yearly or daily or data accumulated per event, the calculator can automatically increment the value of x by 1 for each data point entered. The calculator initially assigns whatever value is in the T-register for the x value of the first data point, then adds 1 for the second, 1 for the third, etc. All data points are entered by pressing 2nd Σ+ only. The starting x value can be set to any number by entering the first x value, then letting the calculator increment from there: $x_1$ x⇄t, $y_1$ 2nd Σ+, $y_2$ 2nd Σ+, $y_3$ 2nd Σ+, etc.

To remove an unwanted data entry, press x⇄t − 1 = x⇄t then enter the unwanted y value.

Example: A computer dating service has the following annual profits:

| Year | 1962 | 1963 | 1964 | 1965-1970 | 1971 | 1972 | 1973 | 1974 |
|---|---|---|---|---|---|---|---|---|
| Profit in millions | −2.1 | −0.3 | 0.8 | inactive | 2.9 | 2.8 | 3.6 | 4.0 |

What profit can be expected in 1980 and when will the company break the $10 million mark?

| Press | Display | Comments |
|---|---|---|
| [2nd] [Pgm] 1 [SBR] [CLR] | 0. | Initialize |
| 1962 [x⇄t] | 0. | Initialize x |
| 2.1 [+/−] [2nd] [Σ+] | 1. | 1962 loss |
| .3 [+/−] [2nd] [Σ+] | 2. | 1963 loss |
| .8 [2nd] [Σ+] | 3. | 1964 gain |
| 1971 [x⇄t] | 1965. | Reinitialize x |
| 2.9 [2nd] [Σ+] | 4. | 1971 gain |
| 2.8 [2nd] [Σ+] | 5. | 1972 gain |
| 3.6 [2nd] [Σ+] | 6. | 1973 gain |
| 4 [2nd] [Σ+] | 7. | 1974 gain |
| 1980 [2nd] [Op] 14 | 6.52181966 | |
| 10 [2nd] [Op] 15 | 1988.297788 | |

In 1980 the company can expect $6.5 million profit and to reach the $10 million mark in 1988.

Any of the statistical capabilities can be used for any of the statistical calculations. Mean, standard deviation, variance, slope and y-intercept could have been calculated in the preceding example.

## STATISTICS IN CALCULATIONS

Statistical operations can be performed during complex calculations. You can have as many as 4 pending operations and still perform statistical calculations. (Statistics requires up to four levels of processing internally.) For instance, some insurance company may compute its overhead by the formula $3 + 2 \times 1.2^{(4+N)}$ where N is the number of people needed for $200,000 sales. Simply key in $3 + 2 \times 1.2^{(4+}$ then with the example from page V-38, calculate x′ for y = 200. After projecting the number of people for $200,000 sales, press [=] to complete the calculation. Notice that four operations are pending while the statistics is being calculated.

Statistical calculations also use one level of subroutine when used in a program. More about subroutines later.

# GENERAL PROGRAMMING

## Programming Your Calculator

To solve a problem from the keyboard, you determine a sequence of operations and functions needed to give you the solution to that problem and key your solution into the calculator. Programming is little more than entering the learn mode and telling the calculator to remember the resulting key sequence. What actually happens is that the keystrokes are stored in locations in *program memory* and each becomes a *program instruction.* The series of keystrokes (instructions) is now a *program.* When the instructions of the program are executed (run) they produce the same result that the equivalent manual keystrokes would have yielded. Once stored, this program can be exercised again and again by supplying new sets of variables instead of entering all the program keystrokes. This not only saves you input time, but decreases the chances of making an entry error, which would further interfere with the problem solving process.

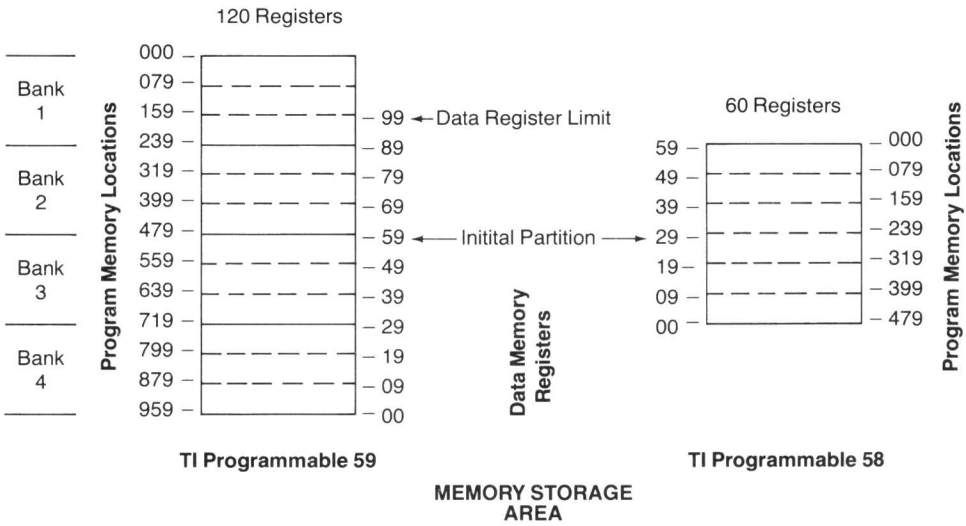
The program stays in program memory until it is replaced by another program, cleared by pressing [2nd] [CP] or the calculator is turned off. Meanwhile, the program can be used whenever you need it. For example, while performing a series of manual operations, you may find you need an answer from a stored program, simply call and execute the program, then return to your calculations with the program results.

This calculator contains a highly sophisticated system for programming — yet it is easy to use. A thorough understanding of the calculator structure will provide you with a problem solving device with capabilities approaching those of a minicomputer.

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

# Storage Capacity and Partitioning

There is a memory storage area within your calculator. This area is for data storage and program storage.

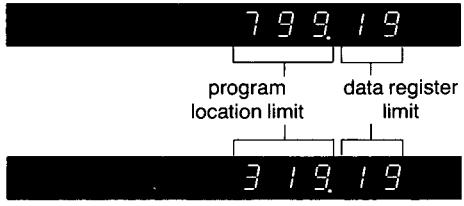



Throughout the remainder of this section, information about the TI Programmable 58 will follow that for the 59 and will be in parentheses in bold numbers.

There are 120 (**60**) registers available for storage in the memory storage area. Initially, there is an even split between program and data memory with 60 (**30**) registers for each. Eight program instructions can be stored in each register of program memory. So you can have a 8 × 60 = 480 (**8 × 30 = 240**) step program with 60 (**30**) registers left for data storage.

You can partition this memory storage area into the ratio you desire in sets of 10 register increments. For example, you can use all 120 (**60**) registers for a program, 120 × 8 = 960 (**60 × 8 = 480**) program locations and no data registers or you can partition so that you have 100 (**60**) data registers and 20 × 8 = 160 (**0**) program locations or other combinations. The only restriction to the system is that a maximum of 100 (**60**) data registers can be used because each memory operation requires a two-digit address, therefore only registers 00-99 can be used.

To partition the storage area, enter the number of sets of 10 data registers you need, 0-10 (**0-6**), and press [2nd] [Op] **17.** Remove all fix-decimal, scientific notation and engineering formats before partitioning. For 20 data registers, press 2 [2nd] [Op] **17** and the display is as follows.




This shows that there are 20 registers, 00-19, available for data storage and 800 (**320**) locations 000-799 (**000-319**), allocated for program storage.

To check the current placement of the partition at any time, press [2nd] [Op] **16** and the existing partition is normally displayed in the format shown above.

## Basic Program Control Functions

Understanding several basic control functions will allow you to begin programming your calculator.

[LRN] — **LEARN** — Pressing this key once puts the calculator in the learn mode of operation. This allows you to begin writing a program into program memory which can be run later. Pressing [LRN] again puts the calculator back under keyboard control and restores the display to its original state. Use of the learn key always preserves the value in the display, but does stop a flashing display if present. You cannot enter the learn mode if a protected program is in program memory.

[2nd] [CP] — **CLEAR PROGRAM** — When pressed from the keyboard, it clears all locations of program memory, clears the subroutine-return register, resets all flags, clears the T-register and resets the program pointer to 000. It also removes program protection. When encountered within a program, it only zeros the T-register.

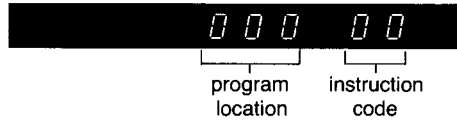
[R/S] — **RUN/STOP** — Reverses the status of processing. If a program memory is running, pressing [R/S] from the keyboard stops it. If it is stopped [R/S] restarts processing at the current position of the program pointer. This allows for data entry or a look at intermediate results before the end of processing.

[RST] — **RESET** — Instructs the calculator to reset the program pointer to location 000 of program memory (even from a library module), clears the subroutine return register and resets all program flags. It is also used to exit from a library program if it seems unable to do so by itself. When [RST] is used for this purpose, all results are lost except those stored in the data registers.

[2nd] [Pause] — **PAUSE** — When encountered during program execution, causes the current value of the display register to be displayed for approximately ½ second. Pause instructions can be used wherever needed, even consecutively. When held down on the keyboard during program execution, it displays the result of each program step. This key is inoperative when running a library program or a protected program.

## Learn Mode

Once a calculation sequence has been determined, select the learn mode by pressing [2nd] [CP] [LRN], then key the sequence into program memory. [2nd] [CP] assures that the program is keyed in beginning at location 000. When you enter the learn mode, the display has the following format.




Program locations begin at 000 and number consecutively up to your partition. Initially, program memory contains zeros in all locations and zeros remain in all locations that are not deliberately changed.

Each location usually contains an operation, an address or just a single digit.

An instruction key code (or just key code) is a two-digit number assigned to each operation according to its actual location on the keyboard. The calculator normally indicates 00 as the key code when a program is being keyed in. This is because, as a complete instruction is entered, the calculator automatically steps to the *next* available location. Instruction key codes are detailed in the section on program editing.

Keeping track of exactly where you are in program memory is the function of the program location pointer (or program pointer). In the learn mode, this indicator moves through program memory displaying the next location to be used.

After keying a program into program memory, press [LRN] again to return the calculator to keyboard control where the variables can be entered and program execution started.

## Entering Your Program

The sequence for keying your program into program memory is:

1. From the keyboard press [RST] or [2nd] [CP]. Either sequence positions the program pointer to 000, the first location of program memory. The [2nd] [CP] sequence also clears program memory.

2. Press [LRN] to place the calculator in the learn mode. The special five-digit display identifies this mode.

3. Key in your program completely: one step at a time, including all necessary [2nd] and [INV] prefixes. The display indicates the *next* location available for an instruction, not the one you just keyed in.

4. Make sure your program did not exceed program memory size. When the last location is filled, the calculator automatically switches to keyboard control where the special display is conspicuously absent.

5. Switch from the learn mode to keyboard control by pressing [LRN] again.

6. Run a test problem with known results to be sure the program is correct and edit your program if necessary.

The following example illustrates the previous comments.

Example: Create a program to calculate the volume of a right circular cylinder of radius "r" and height "h".

Necessary equation: $V = \pi r^2 h$

Desired Program Operation: Enter "r"
Start Program
Halt for "h" entry
Calculate volume, halt and display the answer

| Key Sequence | Display | Comments |
|---|---|---|
| 2nd CP | 0 | Sets program pointer to location 000 and erases program memory |
| LRN | 000 00 | Places calculator in learn mode and begins program entry |
| $x^2$ | 001 00 | $r^2$ (r will be entered before program execution) |
| × | 002 00 | |
| 2nd π | 003 00 | Occupies 1 keystroke |
| × | 004 00 | $\pi r^2$ in display register |
| R/S | 005 00 | Halts for "h" entry |
| = | 006 00 | Calculates result |
| R/S | 007 00 | Stops program, "V" displayed |
| RST | 008 00 | Return to instruction at 000 |
| LRN | 0 | Returns to keyboard control |

The last two keystrokes entered have specific functions. The R/S key halts processing and displays the final answer. The RST key provides a natural return to location 000 when "r" is entered for a new set of variables and R/S is pressed.

The R/S key has the effect of reversing the current status of program execution. If while running a program, the R/S key is either pressed or occurs as an instruction, processing ceases and there is immediate transfer to keyboard control. Then by pressing R/S on the keyboard, the program is restarted and calculations resume from the point where processing was suspended. The program location pointer is left wherever it happened to be at the time of program interruption, thus allowing processing to restart with no adverse effect on calculations, assuming that you do not change anything.

## Running Your Program

When running a program, the instructions are executed in sequential order beginning at the current location of the program pointer. (Advantageous exceptions will be discussed later.) To initiate this processing simply press the R/S (Run/Stop) key. The program pointer keeps up with exactly where processing is in the program. If the calculator attempts to execute past the program boundary, processing halts and the display flashes. Therefore, programs should end with a R/S (or with an INV SBR or a transfer, both to be discussed later).

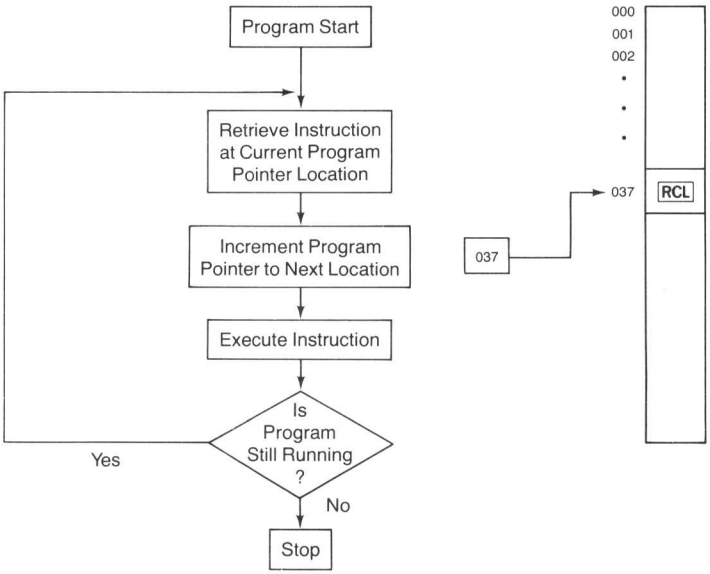
With the volume problem in program memory, let's run the program.

For r = 3 and h = 9, calculate the volume.

| Press | Display | Comments |
|---|---|---|
| [RST] | 0 | Positions program pointer to 000 |
| **3** | 3 | Enter "r" |
| [R/S] | (blank) | Begins program execution |
| | 28.27433388 | $\pi r^2$ — value in display register when [R/S] encountered, halting program |
| **9** | 9 | Enter "h" |
| [R/S] | (blank) | Resumes processing |
| | 254.4690049 | Program halts, displays "V". |

Note that the display is blank (except for a dim "[" in the far left of the display) while a program is being executed. This blank time varies widely depending upon the program being run and the power source being used.

When executing a sequence, the program pointer controls the flow of processing by pointing to each instruction as it is to be executed, as follows.




**Program Location Pointer**

As additional programming capabilities are introduced, the role of the program location pointer will be expanded.

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## Working With Programs

[SST] — **SINGLE-STEP** — Causes the program pointer to be incremented by one. In the learn mode, pressing this key causes the next storage location to be displayed. Pressing this key from the keyboard causes the program to be executed one step at a time with the result of each step being displayed.

[BST] — **BACK-STEP** — In the learn mode, pressing this key causes the program pointer to decrement by one and show the code of the instruction stored there. This key is inoperative from the keyboard and when a program is running.

These keys are inoperative when running library programs.

In the process of keying in a program or looking at it after it is in program memory, two instructions are available that allow movement from instruction to instruction within the program. Pressing the single-step key [SST] in the learn mode increments the program location pointer by one, displaying the next location number and instruction code without affecting the stored instructions in any way. Pressing it repeatedly, therefore allows you to sequentially step through the program memory, observing the codes for the instructions stored there.

Similar to the single-step key, the back-step key [BST] causes the program location pointer to decrement by one location each time it is pressed. A common use of this key is to go back and verify that an instruction just keyed in is the one desired. (You must then press [SST] to go to the next location.) The back-step and single-step instructions combine to provide free movement in program memory to permit you to efficiently check out and "debug" your programs.

The single-step instruction is usable from the keyboard as well as in the learn mode. When pressed while under keyboard control, the effect of [SST] is to cause actual execution of the stored program, one instruction at a time. Each time [SST] is pressed, the instruction located at the current position of the program pointer is executed and the program location pointer is advanced just as if the program had been running. The display shows the calculated values resulting from that instruction. Sometimes several single-step keystrokes are necessary before anything appears to happen, but this is only because the operation in process is a multistep one. For example, the sequence + RCL 09 would take three single-step keystrokes before the recall of the memory register 09 content would actually take place.

As you single-step and back-step through a program while in the learn mode, you see a key code number stored at each location to represent the various instructions.

## Instruction Codes (Key Codes)

In the learn mode, the display shows you where the program location pointer is positioned and the instruction presently residing in that location. The instruction is represented by a two-digit code that usually comes directly from the key's location on the calculator keyboard. The first digit denotes one of the nine rows of keys (numbered 1 through 9 from top to bottom) in which the key is located. The second digit establishes which of the columns contains the key (numbered from left to right 1 through 5). For example, the [STO] key is in row 4 column 2 so its instruction key code is 42. Second functions add 5 to a particular row-column address, i.e., [2nd] [Stn] is code 38 because it is above [$x^2$], code 33. For the far right column of keys, second functions add 5, but do not carry to change the row number. i.e., [2nd] [E'] is code 15 + 5 = 10, not 20. The digit keys 0-9 are designated by the codes 00 through 09.

## Keyboard And Instruction Key Codes

| Key | Key Code | Key | Key Code | Key | Key Code | Key | Key Code | Key | Key Code |
|---|---|---|---|---|---|---|---|---|---|
| A' | 16 | B' | 17 | C' | 18 | D' | 19 | E' | 10 |
| A | 11 | B | 12 | C | 13 | D | 14 | E | 15 |
| | | INV | 27 | log | 28 | CP | 29 | CLR | 20 |
| 2nd | Merged | INV | 22 | ln x | 23 | CE | 24 | CLR | 25 |
| Pgm | 36* | P→R | 37 | sin | 38 | cos | 39 | tan | 30 |
| LRN | None | x≥t | 32 | $x^2$ | 33 | $\sqrt{x}$ | 34 | $1/x$ | 35 |
| Ins | None | CMs | 47 | Exc | 48* | Prd | 49* | Ind | 40 (or merged) |
| SST | None | STO | 42* | RCL | 43* | SUM | 44* | $y^x$ | 45 |
| Del | None | Eng | 57 | Fix | 58* | Int | 59 | \|x\| | 50 |
| BST | None | EE | 52 | ( | 53 | ) | 54 | ÷ | 55 |
| Pause | 66 | x=t | 67* | Nop | 68 | Op | 69* | Deg | 60 |
| GTO | 61* | 7 | 07 | 8 | 08 | 9 | 09 | × | 65 |
| Lbl | 76* | x≥t | 77* | Σ+ | 78 | $\bar{x}$ | 79 | Rad | 70 |
| SBR | 71* | 4 | 04 | 5 | 05 | 6 | 06 | − | 75 |
| St flg | 86* | If flg | 87* | D.MS | 88 | π | 89 | Grad | 80 |
| RST | 81 | 1 | 01 | 2 | 02 | 3 | 03 | + | 85 |
| Write | 96 | Dsz | 97* | Adv | 98 | Prt | 99 | List | 90 |
| R/S | 91 | 0 | 00 | • | 93 | +/− | 94 | = | 95 |

*Keys requiring instructions or addresses to be complete.

Note: The Ind instruction is sometimes numerically merged with the code of the key it is used with.

There is a mylar sheet called a "Key Code Overlay" that comes with your calculator. When placed over the keys, it shows the key code associated with each key.

# The Details



## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

### Key Codes In Numerical Order

| Key Code | Key |
|---|---|
| 00 | 0 |
| ↓ | ↓ |
| 09 | 9 |
| 10 | 2nd E' |
| 11 | A |
| 12 | B |
| 13 | C |
| 14 | D |
| 15 | E |
| 16 | 2nd A' |
| 17 | 2nd B' |
| 18 | 2nd C' |
| 19 | 2nd D' |
| 20 | 2nd CLR |
| 22 | INV |
| 23 | ln x |
| 24 | CE |
| 25 | CLR |
| 27 | 2nd INV |
| 28 | 2nd log |
| 29 | 2nd CP |
| 30 | 2nd tan |
| 32 | x⇄t |
| 33 | $x^2$ |
| 34 | $\sqrt{x}$ |
| 35 | 1/x |
| 36 | 2nd Pgm |
| 37 | 2nd P→R |
| 38 | 2nd sin |
| 39 | 2nd cos |
| 40 | 2nd Ind |
| 42 | STO |
| 43 | RCL |
| 44 | SUM |
| 45 | $y^x$ |
| 47 | 2nd CMs |
| 48 | 2nd Exc |
| 49 | 2nd Prd |
| 50 | 2nd \|x\| |
| 52 | EE |
| 53 | ( |
| 54 | ) |
| 55 | ÷ |
| 57 | 2nd Eng |
| 58 | 2nd Fix |
| 59 | 2nd Int |
| 60 | 2nd Deg |
| 61 | GTO |
| 62 | 2nd Pgm 2nd Ind |
| 63 | 2nd Exc 2nd Ind |
| 64 | 2nd Prd 2nd Ind |
| 65 | × |
| 66 | 2nd Pause |
| 67 | 2nd x=t |
| 68 | 2nd Nop |
| 69 | 2nd Op |
| 70 | 2nd Rad |
| 71 | SBR |
| 72 | STO 2nd Ind |
| 73 | RCL 2nd Ind |
| 74 | SUM 2nd Ind |
| 75 | − |
| 76 | 2nd Lbl |
| 77 | 2nd x≠t |
| 78 | 2nd Σ+ |
| 79 | 2nd $\bar{x}$ |
| 80 | 2nd Grad |
| 81 | RST |
| 83 | GTO 2nd Ind |
| 84 | 2nd Op 2nd Ind |
| 85 | + |
| 86 | 2nd St flg |
| 87 | 2nd If flg |
| 88 | 2nd D.MS |
| 89 | 2nd π |
| 90 | 2nd List |
| 91 | R/S |
| 92 | INV SBR |
| 93 | • |
| 94 | +/− |
| 95 | = |
| 96 | 2nd Write |
| 97 | 2nd Dsz |
| 98 | 2nd Adv |
| 99 | 2nd Prt |

Through normal usage you will become familiar enough with the more common instruction codes that constant reference to these tables will not be necessary. Most key codes can always be quickly determined by reference to the calculator keyboard or the "Key Code Overlay."

## Keystroke Storage

Most instructions occupy one location in program memory. Some instruction sequences do, however, allow several keystrokes to occupy a single location. The [2nd] key is combined with the instruction following it to occupy one location as has been explained. The two-digit addresses accompanying instructions for data memory, program library access and special control operations are combined to occupy one location. For instance, [RCL] **16** occupies only 2 locations as does [2nd] [Pgm] **12** with [2nd] [Pgm] placed in one location and the program number 12 in the next. The calculator automatically does this for you.

Unconditional transfer addresses (discussed on page V-56), like [GTO] **123** are stored [GTO] in one location, 01 in the next and 23 in the next. As you key in this sequence, the calculator assimilates the keystrokes, placing them in the correct locations. No special effort on your part is required.

Sometimes when the indirect key [Ind] is used with another instruction, indirect combines with that instruction to occupy one location, and is assigned a different instruction code. These new codes are not based on the instructions' position on the keyboard, but are assigned key codes of the keyboard locations of the numbers. For example, [STO] [2nd] [Ind] is single-location code 72. These special assignments are handled internally by the calculator and are listed in the numerical order of key codes on the previous page. The indirect instructions affected are:

| Key Sequence | Key Codes |
|---|---|
| [2nd] [Pgm] [2nd] [Ind] | 62 |
| [2nd] [Exc] [2nd] [Ind] | 63 |
| [2nd] [Prd] [2nd] [Ind] | 64 |
| [STO] [2nd] [Ind] | 72 |
| [RCL] [2nd] [Ind] | 73 |
| [SUM] [2nd] [Ind] | 74 |
| [GTO] [2nd] [Ind] | 83 |
| [2nd] [Op] [2nd] [Ind] | 84 |

The indirect instructions that are not affected by this merging simply store the instruction code of [2nd] [Ind] based on its keyboard location, 40, after the code of the instruction it is used with. For instance, [2nd] [x=t] [2nd] [Ind] is stored with [2nd] [x=t], code 67, in one location and [2nd] [Ind], code 40 stored in the next. Uses of indirect sequences are explained in the next section.

## EDITING PROGRAMS

[2nd] [Nop] — **NO OPERATION** — In the learn mode, entry of this key may be used to delete an unwanted instruction or to provide spacing between program parts for later additions. Program execution simply performs no operation when this instruction is encountered. Use of this key does not interfere or alter any key, execution sequence or data entry except when used as a label.

[2nd] [Del] — **DELETE** — In the learn mode, removes the displayed instruction and moves each following instruction up one location so that the gap is eliminated. The program pointer does not move when this key is used.

2nd Ins — **INSERT** — In the learn mode, moves all instructions down one location beginning with the displayed location through to the end of program memory. Used to create room to insert a new instruction.

In the process of working with a program in program memory, you have the following capabilities:

1. Step through the program, forward or backward, displaying the instructions stored in the various locations.
2. Replace any instruction with any other.
3. Delete a displayed instruction and close up the gap.
4. Make a space for a new instruction to be inserted.

These instructions allow you to make corrections or modifications to a program with an absolute minimum amount of effort.

## Replacing an Instruction with Another

A keystroke input at any point in an existing program, while in the learn mode, writes over the instruction previously stored in that location. As you are single-stepping or back-stepping through your program and discover an unwanted instruction, simply press the correct instruction and it will instantly replace the one that was displayed.

## Deleting an Instruction

Any instruction can be removed from the program completely through use of the delete instruction 2nd Del. In the learn mode, a displayed instruction can be deleted and all following instructions are moved up one location so that the gap is removed from the program. As a consequence of closing the gap, a zero is pulled into the last location in program memory. As much of a program can be deleted as necessary.

## Inserting an Instruction

If you need to insert an instruction somewhere in your program, press 2nd Ins to move all instructions beginning with the one displayed down one location creating a hole for a new instruction. Now just key in your new instruction. This operation can be performed repetitively adding sections of code to your program. Each time this option is used, though, the instruction in the last available location of program memory is lost.

A no-operation instruct on 2nd Nop is available to delete an instruction from a location and leave a no-op code in its place. This code is ignored when encountered in the execute mode except when it is used as a label. Use of this key can reserve a particular point in a program that is subject to frequent change. This makes program alteration possible without using the delete and insert options that change program locations.

Example: Key in and edit $x^2 + 3x - 2$ to be solved several times for different values of x. Assume x to be in the display when program execution is started.

| Press | Display | Comments |
|---|---|---|
| LRN | 000 00 | Enter learn mode |
| STO | 001 00 | Store x for later use |
| 1 | 001 00 | |
| $x^2$ | 003 00 | |
| − | 004 00 | Incorrect entry |
| BST | 003 75 | Back step to 003 |
| + | 004 00 | Replace − with + |
| 4 | 005 00 | Incorrect entry |
| × | 006 00 | |
| BST | 005 65 | Back step to incorrect entry |
| BST | 004 04 | |
| 3 | 005 85 | Replace 4 with 3 |
| SST | 006 00 | Single step past correct × |
| × | 007 00 | |
| RCL | 008 00 | |
| 1 | 008 00 | |
| BST | 008 01 | Discover × entered twice |
| BST | 007 43 | Back step to location 006 |
| BST | 006 65 | |
| 2nd Del | 006 43 | Deletes ×, bring RCL from 007 to 006, etc. |
| SST | 007 01 | Single step past RCL |
| SST | 008 00 | Single step past 1 |
| 2 | 009 00 | |
| = | 010 00 | |
| BST | 009 95 | Realize − missed |
| BST | 008 02 | Back step to location 008 |
| 2nd Ins | 008 00 | Vacates 008 moving 2 = down |
| − | 009 02 | Insert − |
| SST | 010 95 | Single step past 2 |
| SST | 011 00 | Single step past = |
| R/S | 012 00 | Halts program, displays answer |
| RST | 013 00 | |

Now to verify that the program has been entered correctly, single-step through the program. The complete program should be as follows.

| Location and Key Code | | Key Sequence |
|---|---|---|
| 000 | 42 | STO |
| 001 | 01 | 0 1 |
| 002 | 33 | $x^2$ |
| 003 | 85 | + |
| 004 | 03 | 3 |
| 005 | 65 | × |
| 006 | 43 | RCL |
| 007 | 01 | 0 1 |
| 008 | 75 | − |
| 009 | 02 | 2 |
| 010 | 95 | = |
| 011 | 91 | R/S |
| 012 | 81 | RST |

Be very careful of what you edit because of merged key codes. Consider the following sequence.

| Location and Key Code | | Key Sequence |
|---|---|---|
| • | | |
| • | | |
| • | | |
| 019 | 95 | = |
| 020 | 42 | STO |
| 021 | 12 | 1 2 |
| 022 | 61 | GTO |
| • | | |
| • | | |
| • | | |

If the STO is deleted, the register number 12 now becomes B (code 12). If the 12 needs to be replaced with 13, positioning the pointer to 021 and entering 13 places the 1 in 021 and the 3 in 022 not 13 in 021. To achieve this change you can either enter STO **13** from location 020 on or enter a C (code 13) at location 021. This clever entry of C to get code 13 in 021 is a very useful editing technique for altering program instructions.

# LABELING PROGRAM PARTS

[2nd] [Lbl] — **LABEL** — Used only in the learn mode to label segments of a program. This key instructs the calculator to use the next keyboard entry as a marker and not as an instruction to be performed.

Using labels is similar to using tabs in a notebook. These are especially useful for programs with more than one part that you can run separately or sequentially. There are two types of labels, user-defined labels and common labels.

## User-Defined Keys as Labels

A series of keys at the top of the keyboard, [A] – [E] and [2nd] [A'] – [2nd] [E'], are called user-defined keys. Labeling a segment of your program with one of these keys allows you to press that key in the calculate mode and have that program segment accessed instantly. You actually define the function of that key so that it acts like the other functions on the keyboard. The sequence may completely fill program memory or can be short as in the previous example. If that example is still in your calculator, give the program a user-defined label.

| Press | Display | Comments |
|---|---|---|
| [LRN] | 0 | Return to calculate mode |
| [RST] | 0 | Return to 000 |
| [LRN] | 000 42 | Enter learn mode |
| [2nd] [Ins] | 000 00 | Move program down 2 locations |
| [2nd] [Ins] | 000 00 | |
| [2nd] [Lbl] | 001 42 | Place [Lbl] in 000 |
| [A] | 002 42 | Place [A] in 001 |

Now the program is labeled A and can be run by entering an x value into the display and pressing [A].

| | | |
|---|---|---|
| 1 [A] | 2. | Expression evaluated for x = 1 |
| 1 [+/–] [A] | –4. | For x = –1 |
| 6 [A] | 52 | For x = 6 |
| 3.2 [EE] | 3.2 00 | |
| 12 [A] | 1.024 25 | For x = $3.2 \times 10^{12}$ |
| [2nd] [π] [÷] | 3.1415927 00 | |
| .03 [=] [A] | 1.1278386 04 | For x = $\pi/.03$ |

Each of the user-defined keys can be assigned to a program sequence and executed at will. Whether a user-defined key is pressed from the keyboard or encountered while running a program, relocation is made to the area of the program labeled with that user-defined key and processing is performed.

# The Details

# V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

## Common Labels

Almost any key on the keyboard (including second functions) may be used as a label. Only the following keys cannot be used as labels: [2nd], [LRN], [Ins], [Del], [SST], [BST], [Ind] and the numbers 0-9. You should also avoid using [R/S] as a label because of its ability to start program execution.

These labels are assigned just as are the user-defined labels. But, program segments tagged with common labels cannot be accessed quite as easily from the keyboard as are the user-defined labels. You can have a segment labeled cos, for instance. Pressing [2nd] [cos] on the keyboard is an executable command so the cosine of the display is taken. Common labels are primarily designed for use within programs for transfer purposes although they can be accessed from the keyboard.

Pressing [GTO] [2nd] [cos], for instance, sends the program pointer to the instruction immediately following label "cos." [SBR] [2nd] [cos] does the same, and also starts processing at that point as would pressing [GTO] [2nd] [cos] [R/S].

In general, you can use as many different labels as you need (there are 72 available) in a program. No label can be used to label more than one program segment, but any label can be called as often as necessary.

All labels including their key codes that are used in the current program memory can be listed on the PC-100A Printer along with the locations in program memory. Simply press [GTO] **0** or [RST] (to position the program pointer to location 000), then press [2nd] [Op] **08** and the table is printed out.

## TRANSFER INSTRUCTIONS

Up to this point, when a program is executed, processing begins at 000 and progresses sequentially until a Run/Stop instruction is encountered. There are a series of keys called transfer instructions that allow you to interrupt this consecutive flow of processing. There are two types of transfers — unconditional and conditional. The unconditional transfer instructions immediately relocate the program pointer to the requested location. Conditional transfers perform a test on the program status and transfer only under certain conditions.

## Unconditional Transfer Instructions [GTO] and [SBR]

### GO TO INSTRUCTION

[GTO] **N** or **nnn — GO TO INSTRUCTION** — When used in a program, Go To instantly diverts the flow of processing to label **N** or program location **nnn. N** can either be a user-defined label or a common label. From the keyboard, Go To positions the program location pointer to the part of the program labeled **N** or to location **nnn**, but does not start program execution. The keyboard use of this key is useful for rapid access to any part of a program for editing or other purposes.

Transferring to a specific program location (**nnn**) is called *absolute addressing.*

Example: Write a program to count by fours.

| Key Sequence | Comments |
|---|---|
| [2nd] [CP] | Clear program memory, reset to location 000 |
| [LRN] | Enter learn mode |
| [2nd] [Lbl] [SUM] | Program segment labeled SUM |
| [+] [4] [=] | Adds 4 to each result |
| [2nd] [Pause] | Pauses and displays number |
| [GTO] [SUM] | Goes to label SUM and the sequence repeats |
| [LRN] | Exit learn mode |
| [GTO] [SUM] | Pointer directed to label SUM from the keyboard. |

Now simply press [R/S] and you'll see 4, then 8, then 12, etc. The calculator performs the sequence +4 = over and over until you press [R/S] again.

This program could have been done several ways. Let's assume that each sequence begins at location 000.

| Key Sequence | Key Sequence | Key Sequence | Key Sequence |
|---|---|---|---|
| [2nd] [Lbl] | [+] | [2nd] [Lbl] | [+] |
| [SUM] | [4] | [C] | [4] |
| [+] | [=] | [+] | [=] |
| [4] | [2nd] [Pause] | [4] | [2nd] [Pause] |
| [=] | [GTO] | [=] | [RST] |
| [2nd] [Pause] | [0] | [2nd] [Pause] | |
| [GTO] | [0] | [C] | |
| [SUM] | [0] | | |
| **Common Label** | **Absolute Addressing** | **User-Defined Label** | **Reset** |

The common label method is the one used initially. You can see that when you use absolute addressing, no labels are needed. But, if this segment was lower in program memory and editing occurred at some lesser location number, the absolute address would have to be changed. The user-defined label transfers to Label C and automatically performs the calculation. This method does fill up the subroutine return register discussed below.

A three-digit absolute address is merged into two locations when it is stored. For instance, [GTO] **126** is stored in three locations GTO 01 26. If you are working with addresses less than 100 only the significant part of an address need be entered. Pressing [GTO] **7** is the *short-form* address which automatically stores the sequence in the proper three locations GTO 00 07 as soon as a **nonnumeric** key is pressed. But, at least one digit must always be submitted or GTO will accept the next entry as a label.

## SUBROUTINES

[SBR] **N** or **nnn — SUBROUTINE** — A *subroutine* is a sequence of instructions which can be written to define a mathematical or logical operation separate from the main portion of your program. The main program or another subroutine can at any time call and execute this sequence. Subroutines are designed for program situations that contain one or more series of steps that are used over and over. Instead of having to write the series each time it is needed, it can be written once and called whenever necessary. After a subroutine has been called and its function completed, control is returned to the program address following the subroutine calling sequence.

In the learn mode [SBR] programs a transfer to a subroutine labeled **N** or location **nnn.** When encountered while the program is running, the flow of processing is immediately diverted to the subroutine called. The location number following the subroutine call is stored in the *subroutine return register.* [INV] [SBR] terminates each subroutine and processing transfers back to the location stored in the subroutine return register where processing continues. [INV] [SBR] acts like a Run/Stop if not used in a subroutine. The subroutine return register can hold six return locations allowing one subroutine to call another. Each time a subroutine is completed, the associated return address is removed from the subroutine return register, making room for another subroutine if needed.

When this instruction is used from the keyboard, the program transfers to the requested label or location and processing begins automatically.

Example: Evaluate $x^2 - 3x - 2$ for $x_1$ stored in $R_{01}$ and $x_2$ stored in $R_{02}$ and sum the two results into $R_{03}$.

Because $x^2 - 3x - 2$ is to be evaluated twice, write a subroutine to perform this task. Arbitrarily begin at location 030 by pressing [GTO] **30** [LRN] and enter the following:

| Location and Key Codes | Key Sequence | Comments |
|---|---|---|
| 030 76 | [2nd] [Lbl] | Program segment labeled CE |
| 031 24 | [CE] | |
| 032 53 | [(] | |
| 033 42 | [STO] | Store incoming x for later use |
| 034 05 | [5] | |
| 035 33 | [$x^2$] | |
| 036 75 | [−] | |
| 037 03 | [3] | |
| 038 65 | [×] | |
| 039 43 | [RCL] | Recall x |
| 040 05 | [5] | |
| 041 75 | [−] | |
| 042 02 | [2] | |
| 043 54 | [)] | Evaluates expression |
| 044 44 | [SUM] | Sums results into register 03 |
| 045 03 | [3] | |
| 046 92 | [INV] [SBR] | Returns to main program |

Note that parentheses were used here to evaluate the expression. An equals could have been used, but it would have completed any pending operations in the main routine as well as these in the subroutine. Be safe and use parentheses. Now the main program can be written to input the "x" values in the proper place and display the results.

| Location and Key Codes | Key Sequence | Comments |
|---|---|---|
| 000 43 | RCL | Recall $x_1$ |
| 001 01 | 1 | |
| 002 71 | SBR | Call subroutine CE |
| 003 24 | CE | |
| 004 43 | RCL | Recall $x_2$ |
| 005 02 | 2 | |
| 006 71 | SBR | Recall subroutine CE |
| 007 24 | CE | |
| 008 43 | RCL | Recall answer in register 03 |
| 009 03 | 3 | |
| 010 91 | R/S | Halts and display $R_{03}$ |
| 011 81 | RST | Resets to 000 for next variables also zeros subroutine return register |

When the subroutine is called, processing branches to label CE (location 030), completes the subroutine and returns to program location 004. The next subroutine returns to location 008. Processing always returns from the subroutine to the program location immediately following the subroutine instruction that called that subroutine.

Additionally, subroutines called by the main program can themselves call additional subroutines. The subroutine return register automatically keeps track of where each subroutine is to return upon completion.

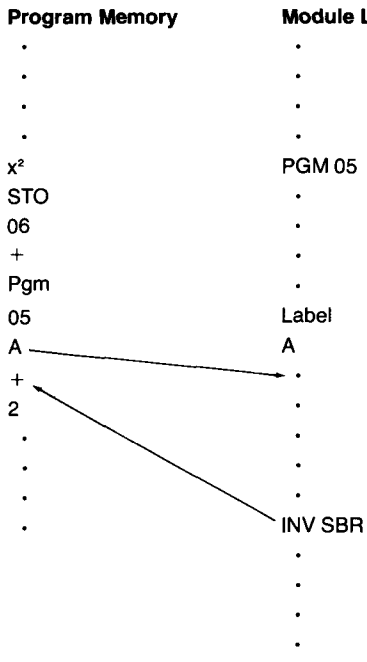
After a program has been keyed in, pressing RST to return to location 000 also zeros the subroutine return register.

## LIBRARY PROGRAMS AS SUBROUTINES

Any program on a library module in place in the calculator can be called as a subroutine. The programs on each module are designed specifically for this purpose with each part of every program being a subroutine. Each part is labeled, ends in INV SBR and most contain no Run/Stop instructions. See the library manuals for the labeling of each program part.

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

You have seen how to access library programs from the keyboard. When running a program in program memory, you can call a library program and use any of its parts as a subroutine. If the part is labeled with a user-defined key, the program sequence [2nd] [Pgm] **mm, N** in program memory transfers to library program **mm** and continues program execution in the library program at label **N.** A common label in a library program is accessed by [2nd] [Pgm] **mm** [SBR] **N.** When the library program segment labeled **N** has completed its purpose, the program location pointer reverts back to the instruction immediately following the original point of departure from program memory and processing continues.




When encountered in a program, the instruction [2nd] [Pgm] **mm, N** behaves just like [SBR] **N** in that the address following this instruction is stored in the subroutine return register and processing immediately diverts to the section of the program labeled **N.** As soon as the sequence following label **N** is completed, processing returns to the last address stored in the return register. Following [2nd] [Pgm] **mm** with anything other than [SBR] or a user-defined key is not a valid key sequence and can cause unwanted results.

# The Details

V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

Use of the library programs as subroutines greatly increases the calculating capacity of your calculator. There are some programming aspects that become critical when any two separate programs are brought together. The programs must not interfere with each other by attempting to use the same data memory registers or flags. A library program can also call other subroutines so you need to be sure that no more than 6 subroutine levels are used. You are usually safe if you use no more than three subroutine levels in program memory. Also, the display format of a library program could be in fix-decimal or scientific or engineering notation. The various uses of subroutines are briefly summarized below.

**SUBROUTINE INSTRUCTIONS**

| From | To | Key Sequence |
|---|---|---|
| Program Memory or from one Library Program to another | Program Library | [2nd] [Pgm] **mm, N** (user defined label) or [2nd] [Pgm] **mm** [SBR] **N** (common label) or [2nd] [Pgm] **mm** [R/S] |
| Program Memory | Program Memory | [SBR] **nnn** or **N** |
| Program Library | Program Memory | [2nd] [Pgm] **00, N** (or **nnn**) or [2nd] [Pgm] **00** [SBR] **N** or [RST] |

## Conditional Transfers (Test Instructions)

These instructions transfer program operation only if the value in the display register satisfies some specific condition: what is the status of the current value (stored in the display register) in relation to some other value (stored in the T-register mentioned earlier) or has processing been through enough steps (decrement and skip on zero instruction?).

### T-REGISTER COMPARISONS

| | |
|---|---|
| [x⇄t] | Exchanges display register value x with T-register value t. |
| [2nd] [x=t] **N** or **nnn** | Asks "Is the display register value exactly equal to the T-register value?" |
| [INV] [2nd] [x=t] **N** or **nnn** | Asks "Is the display register value unequal to the T-register value?" |
| [2nd] [x≥t] **N** or **nnn** | Asks "Is the display register value greater than or exactly equal to the T-register value?" |
| [INV] [2nd] [x≥t] **N** or **nnn** | Asks "Is the display register value less than the T-register value?" |

When the answer is "yes" to any of the above questions the flow of processing branches to the address that immediately follows the instruction. If the answer is "no", processing skips the accompanying address and goes on to the next instruction.

These tests do not affect pending operations, hence they can be used wherever desired in a program.

Example:

| Location and Key Code | Key Sequence |
|---|---|
| 022 43 | RCL |
| 023 01 | 1 |
| 024 95 | = |
| 025 77 | 2nd x≠t |
| 026 12 | B |
| 027 61 | GTO |
| 028 00 | 3 |
| 029 03 | |
| 030 76 | 2nd Lbl |
| 031 12 | B |
| 032 91 | R/S |

In this sequence, the result produced at location 024 is tested in location 025 to see if it is greater than or equal to the value stored in the T-register. If the answer to the test is yes, the flow of processing jumps to label B where processing stops and the result is displayed. If the answer is no, the transfer to label B is skipped and GTO 003 transfers to location 003 where processing continues.

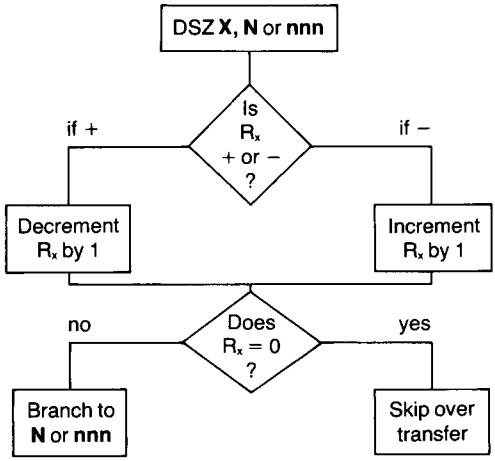
## DECREMENT AND SKIP ON ZERO (DSZ)

2nd DSZ **X, N** or **nnn — DECREMENT AND SKIP ON ZERO** — Decreases the magnitude of the contents of data register X (0-9) by one and processing transfers to label **N** or location **nnn** when $R_x \neq 0$. The transfer is skipped when $R_x$ is zero. $R_x$ here represents the contents of X.

NOTE: Conditional branches do not store a return address in the subroutine return register. If you need to call a subroutine and return on a conditional test, make the test the first step in the subroutine. For instance, 2nd x≠t x² does not store a return address, but SBR x² then 2nd Lbl x² 2nd x≠t does store a return address.

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

This powerful programming instruction is an effective counter as well as a test instruction. If you need to iterate (repeat) a sequence y number of times, just store y in a data register (0-9) and program a DSZ test into the iterative sequence. After y iterations the looping ceases and the program continues. The DSZ instruction operates as follows:




Instruction encountered in program

Is $R_x$ + or −?

Reduce magnitude of $R_x$ by one

Does $R_x = 0$?

If $R_x$ is not 0, transfer to **N** or **nnn**
If $R_x$ is 0, skip the transfer and continue processing.

Understanding of this illustration shows that if you place the DSZ instruction at the beginning of a sequence, it counts then performs the calculation sequence. If placed at the end of the sequence, the function is performed then the count is made. All this means is that to obtain the correct number of passes, y, through a sequence, enter y into register X initially and perform DSZ last or perform DSZ first, but initially store y + 1 in register X.

Example: Write a program to calculate F! (F-factorial) where F! = 1 × 2 × 3 × . . . ×F. (0! = 1 by definition)

| Location and Key Codes | Key Sequence | Comments |
|---|---|---|
| 000 76 | 2nd Lbl | |
| 001 15 | E | |
| 002 42 | STO | Store F in register 00 |
| 003 00 | 0 | |
| 004 29 | 2nd CP | Zero T-register |
| 005 67 | 2nd x=t | Test F = 0? |
| 006 11 | A | If yes, transfer to A |
| 007 76 | 2nd Lbl | |
| 008 12 | B | |
| 009 43 | RCL | Recall F |
| 010 00 | 0 | |
| 011 65 | × | |
| 012 97 | 2nd Dsz | Decrement $R_{00}$ by 1 |
| 013 00 | 0 | |
| 014 12 | B | If $R_{00}$ is not within ±1, transfer to B |
| 015 76 | 2nd Lbl | If $R_{00}$ is within ±1, proceed to the end |
| 016 11 | A | |
| 017 01 | 1 | |
| 018 95 | = | |
| 019 91 | R/S | Stops and displays F! |

To execute this sequence, simply enter an F number and press E. F must be less than 70 or the calculator overflows because 70! > 9.9999999 × $10^{99}$.

FLAGS

2nd St flg **y — SET FLAG** — Raises or turns on flag number **y** where 0 ≤ **y** ≤ 9. INV 2nd St flg **y** lowers or zeros flag **y**, resets flag **y** to 0.

2nd If flg **y, N** or **nnn — FLAG SET TEST** — Tests flag **y** to see if it is raised (set). If so, transfer is made to label **N** or location **nnn.**

INV 2nd If flg **y, N** or **nnn — FLAG NOT SET TEST** — Tests flag **y** and transfer to address **N** or **nnn** if **y** is not set.

There are ten flags numbered 0 through 9 available for use in your programs. These flags are initially lowered (set to zero) and can be used to track which path the execution of a program took to completion or to control program options from the keyboard prior to execution. All flags are set to zero and the subroutine register is cleared when [RST] or [2nd] [CP] is pressed from the keyboard. The RST instruction in a program also zeros all flags.

Example: Design a program sequence to sum all incoming numbers, but print only the positive ones and display the sum after each entry.

| Location and Key Code | Key Sequence | Comments |
|---|---|---|
| 018 76 | [2nd] [Lbl] | |
| 019 11 | [A] | |
| 020 22 | [INV] | |
| 021 22 | [2nd] [St flg] | |
| 022 86 | [3] | |
| 023 03 | [2nd] [x≥t] | "Is number nonnegative?" |
| 024 12 | [B] | If so, go to label B |
| 025 86 | [2nd] [St flg] | If negative, set flag 3 |
| 026 03 | [3] | |
| 027 76 | [2nd] [Lbl] | |
| 028 12 | [B] | |
| 029 44 | [SUM] | Sum all numbers |
| 030 12 | [1] [2] | |
| 031 87 | [2nd] [If flg] | "Is flag 3 set?" |
| 032 03 | [3] | |
| 033 13 | [C] | If yes (if number is negative), go to C |
| 034 99 | [2nd] [Prt] | If not, print number |
| 035 76 | [2nd] [Lbl] | |
| 036 13 | [C] | |
| 037 43 | [RCL] | |
| 038 12 | [1] [2] | |
| 039 91 | [R/S] | |

Be sure that data register 12 is clear before entering a series of numbers. Entering a number and pressing [A] sums that number into data register 12, prints entry if it is positive and displays the total of all entries.

## FLAGS AND ERROR CONDITIONS

Flag 8 has been earmarked to determine program operation according to the error condition status of a program. Normally, a program continues running even though an error condition has occurred. If flag 8 is set either from the keyboard or in a program, program execution is suspended when an error condition occurs.

2nd Op **18** says that if no error condition exists in a program, set flag 7.
2nd Op **19** says that if an error condition does exist in a program, set flag 7. Flag 7 can now be monitored to determine the error status of your program and appropriate responses can then be made.

If either of these tests is false, flag 7 is not altered.

# The Details

V

## AN IN-DEPTH ANALYSIS OF FEATURES AND FUNCTIONS

### Indirect Addressing

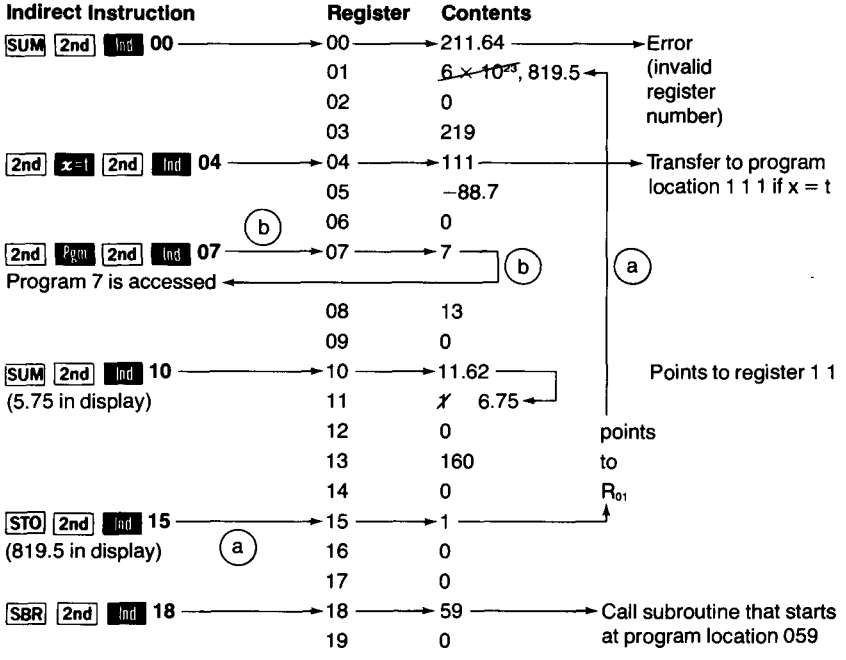
[2nd] [Ind] **XX — INDIRECT SUFFIX** — When used after one of the following operations, recalls the contents of data register **XX** and uses these as the correct address for transferring or for the actual data register to use in processing.

| Key Sequence | Key Codes | Purpose |
|---|---|---|
| [STO] [2nd] [Ind] **XX** | 72 XX | Indirect store |
| [RCL] [2nd] [Ind] **XX** | 73 XX | Indirect recall |
| [2nd] [Exc] [2nd] [Ind] **XX** | 63 XX | Indirect exchange |
| [SUM] [2nd] [Ind] **XX** | 74 XX | Indirect sum to memory |
| [INV] [SUM] [2nd] [Ind] **XX** | 22 74 XX | Indirect subtract from memory |
| [2nd] [Prd] [2nd] [Ind] **XX** | 64 XX | Indirect multiply into memory |
| [INV] [2nd] [Prd] [2nd] [Ind] **XX** | 22 64 XX | Indirect divide into memory |
| [GTO] [2nd] [Ind] **XX** | 83 XX | Indirect go to |
| [2nd] [Pgm] [2nd] [Ind] **XX** | 62 XX | Indirect program call |
| [2nd] [Op] [2nd] [Ind] **XX** | 84 XX | Indirect special control |
| [SBR] [2nd] [Ind] **XX** | 71 40 XX | Indirect subroutine |
| [2nd] [x≥t] [2nd] [Ind] **XX** | 77 40 XX | Indirect x≥t test |
| [INV] [2nd] [x≥t] [2nd] [Ind] **XX** | 22 77 40 XX | Indirect x<t test |
| [2nd] [x=t] [2nd] [Ind] **XX** | 67 40 XX | Indirect x = t test |
| [INV] [2nd] [x=t] [2nd] [Ind] **XX** | 22 67 40 XX | Indirect x ≠ t test |
| [2nd] [Fix] [2nd] [Ind] **XX** | 58 40 XX | Indirect fix-decimal |
| [2nd] [St flg] [2nd] [Ind] **XX** | 86 40 XX | Indirect flag set |
| [INV] [2nd] [St flg] [2nd] [Ind] **XX** | 22 86 40 XX | Indirect flag reset |
| [2nd] [Dsz] [2nd] [Ind] **XX, N** or **nnn** | 97 40 XX | Indirect register DSZ test |
| [2nd] [Dsz] **X** [2nd] [Ind] **XX** | 97 X 40 XX | Indirect register DSZ test |
| [2nd] [Dsz] [2nd] [Ind] **XX** [2nd] [Ind] **yy** | 97 40 XX 40 yy | Indirect register and address, DSZ test |
| [INV] [2nd] [Dsz] [2nd] [Ind] **XX, N** or **nnn** | 22 97 40 XX | Indirect register, skip on nonzero test |
| [INV] [2nd] [Dsz] **X** [2nd] [Ind] **XX** | 22 87 X 40 XX | Indirect address, skip on nonzero test |
| [INV] [2nd] [If flg] [2nd] [Ind] **XX** [2nd] [Ind] **yy** | 22 87 40 XX 40 yy | Indirect address and address, skip on nonzero test |
| [2nd] [If flg] [2nd] [Ind] **XX, N** or **nnn** | 87 40 XX | Indirect flag number, flag set test |
| [2nd] [If flg] **X** [2nd] [Ind] **yy** | 87 X 40 yy | Indirect address, flag set test. |
| [2nd] [If flg] [2nd] [Ind] **XX** [2nd] [Ind] **yy** | 87 40 XX 40 yy | Indirect flag number and address, |
| [INV] [2nd] [If flg] [2nd] [Ind] **XX, N** or **nnn** | 22 87 40 XX | Indirect flag number, flag not set test |
| [INV] [2nd] [If flg] **X** [2nd] [Ind] **XX** | 22 87 X 40 XX | Indirect address, flag not set test. |
| [INV] [2nd] [If flg] [2nd] [Ind] **XX** [2nd] [Ind] **yy** | 22 87 40 XX 40 yy | Indirect flag number and address, flag not set test |

Notice that the indirect memory operations, PGM, Op and Go To instructions have assigned codes that do not represent their positions on the keyboard. These codes have been assigned to save program locations.

The indirect instruction produces a transfer of memory address by finding it stored in data register XX. The effect of pressing [RCL] [2nd] [Ind] **04** would not be to recall the contents of register 04, but to use $R_{04}$ as the requested memory address. If 111 is stored in register 04 then 111 becomes the transfer address. The indirect address "points" to the actual address.

The diagram below illustrates these concepts graphically.




This diagram shows the effect of [2nd] [x=t] [2nd] [Ind] **04**. The result is to use the value in $R_{04}$ (211) as the transfer address for the x = t instruction. The effect of [STO] [2nd] [Ind] **15** is shown as the path of events marked (a). The result is to store 819.5 into $R_{01}$, formerly containing $6 \times 10^{23}$. Finally, the result of [2nd] [Pgm] [2nd] [Ind] **07** in this example is marked (b). The pointer points back to data register 07 so that the result is to use the value 7 to access library program number 07.

It is implied that any pointer used as the consequence of an indirect instruction must point to a realizable address. Accordingly, the result of the [SUM] [2nd] [Ind] **00** in the example diagram would be an error — there is no data register 211.64. [SUM] [2nd] [Ind] **10** sums 5.75 into register 11, because only the integer portion of an indirect address is used.

If the value in the indirect register is less than zero, the calculator uses register 00. If the value is beyond the partition, processing halts and the display flashes.

# VI PRINTER CONTROL

The optional **PC-100A Print/Security Cradle*** can be used with your programmable calculator to perform a number of different printing tasks with the calculator in place on the printer, you can:

1. Print the contents of the display at any time.
2. List your program in program memory and the contents of all data registers.
3. Print results from any point of a running program.
4. Print each step of calculator operation by tracing calculations made from the keyboard or in a program.
5. List all program labels and the program location of each.
6. Print alphanumeric messages wherever needed.
7. Make a plot of data from the keyboard or automatically from a program.
8. Leave your desk without having to lock your calculator away. The printing unit provides security as well as power for your calculator.
9. The calculator's rechargeable battery can be charging while the printer is in use.

Operating instructions for this versatile printer are included with the unit itself, but there are several additional things you need to know.

The calculator selection switch located in the storage/charging compartment of the printer should be set to "OTHER" for use with either the TI Programmable 58 or 59.

The audit trail symbols and the program for cleaning the print head are different for these calculators and are discussed later.

*Note: The PC-100 Print Cradle does not operate with the TI Programmable 58 and 59, only the PC-100A.

## SELECTIVE PRINTING

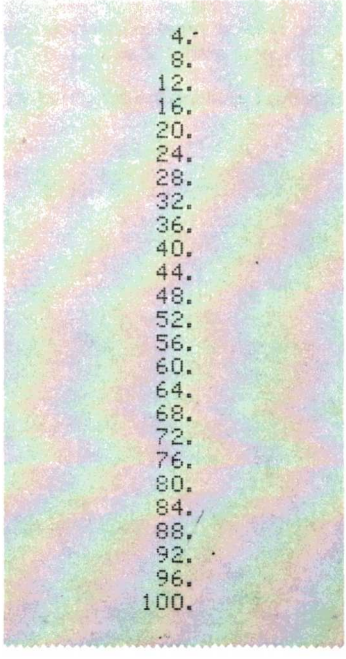
Whenever [2nd] [Prt] is pressed from the keyboard or encountered in a program, the display value is printed. Pressing the [PRINT] key on the printer is the same as pressing [2nd] [Prt] from the keyboard. If an error condition exists when these instructions are encountered, a question mark is printed to the right of the value printed.

Consider the following program that prints multiples of 4:

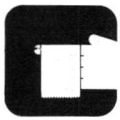
| Location and Key Code | Key Sequence |
|---|---|
| 000 85 | [+] |
| 001 04 | [4] |
| 002 95 | [=] |
| 003 99 | [2nd] [Prt] |
| 004 81 | [RST] |

Enter some starting point and press [RST] [R/S] and the following is obtained.

Results



# VI



## Printer Control

A special set of print instructions are in the library module, so if a PC-100A printer is available, the entries and results from library programs can be automatically printed.

Segments of printed results can be separated by using the paper advance key [2nd] [Adv] on the calculator. When used from the keyboard or in a program, it advances the paper one blank space without printing. For instance, to separate our multiples of 4, insert [2nd] [Adv] into any location ahead of [RST] and you get the following. This instruction has no effect on calculations.

Results

```
4.

8.

12.

16.

20.

24.

28.

32.

36.

40.

44.

48.

52.
```

To obtain additional space, simply repeat the instruction several times.

The [ADV↑] key on the printer can also create blank space. It continually advances the paper as long as the key is held down.

## LISTING YOUR PROGRAM

To list a program, simply press [2nd] [List] from the keyboard. The program is then listed from the current position of the program pointer, to the end of program memory. You can manually stop the listing at any time by pressing [R/S]. For a complete program listing, press [RST] [2nd] [List]. The multiples of 4 program lists like this:

| Program Location | Key Code | Key Symbol |
|---|---|---|
| 000 | 85 | + |
| 001 | 04 | 4 |
| 002 | 95 | = |
| 003 | 99 | PRT |
| 004 | 98 | ADV |
| 005 | 81 | RST |
| 006 | 00 | 0 |
| 007 | 00 | 0 |

## LISTING DATA REGISTERS

The sequence [INV] [2nd] [List] from the keyboard or in a program lists the contents of all data registers beginning with the register number shown in the display. The listing continues until the contents of the highest numbered data registers are listed or until you press [R/S] from the keyboard. Now the calculator is back under keyboard control. A listing of the arbitrary contents of registers 50 up to the partition at 59 is shown below.

| Register Contents | Register Number |
|---|---|
| 14.18181818 | 50 |
| 665.8568182 | 51 |
| 110.9761364 | 52 |
| 0. | 53 |
| 0. | 54 |
| -5.5488068-12 | 55 |
| 0. | 56 |
| 0. | 57 |
| 1. | 58 |
| 0.085106383 | 59 |

This listing was obtained by pressing **50** [INV] [2nd] [List].

When this option is used in a program calculations cease, the requested data register contents are listed and the program stops, returning control to the keyboard.

## TRACING YOUR CALCULATIONS

By pressing [TRACE] on the printer, you cause every step of a calculation to be printed. The calculated value and the instruction that created it are displayed. This is true for both keyboard calculations and program calculations.

The [TRACE] key is a latching switch which in the down position causes trace mode operation for all calculations. In this mode every new function or result is automatically printed. A number entry is only printed if followed by an operation or function. Operation in the trace mode continues until the [TRACE] key is pressed again to release it. When an error condition occurs, a question mark is printed to the right of the value printed.

With the "multiples of 4" program still in program memory, press the [TRACE] key on the printer. Now, press [CLR] [RST] [R/S] on the calculator to obtain the following "tracing" of the calculations that take place.

| Display Register | Audit Symbols |
|---|---|
| 0. | + |
| 4. | = |
| 4. | |
| 4. | PRT |
| | |
| | RST |
| 4. | + |
| 4. | = |
| 8. | |
| 8. | PRT |
| | |
| | RST |
| 8. | + |
| 4. | = |
| 12. | |
| 12. | PRT |
| | |
| | RST |
| 12. | + |
| 4. | = |
| 16. | |
| 16. | PRT |

The program (and consequently the tracing) was stopped by pressing [R/S].

## AUDIT TRAIL SYMBOLS IN TRACE MODE

Most of the symbols listed by the printer are easily identified while others are somewhat obscure. A complete list of all audit symbols and the key sequence that created each one follows.

# Printer Control



| Printer Listing | Key Sequence |
|---|---|
| A – E | A – E |
| A′ – E′ | 2nd A′ – 2nd E′ |
| ADV | 2nd Adv |
| BST | See Note Below |
| CE | CE |
| CLR | CLR |
| CP | 2nd CP |
| CMS | 2nd CMs |
| COS | 2nd cos |
| DEG | 2nd Deg |
| DEL | See Note Below |
| DMS | 2nd D.MS |
| DSZ | 2nd Dsz |
| EE | EE |
| ENG | 2nd Eng |
| EQ | 2nd x=t |
| EX* | 2nd Exc 2nd Ind |
| EXC | 2nd Exc |
| FIX | 2nd Fix |
| GE | 2nd x≥t |
| GO* | GTO 2nd Ind |
| GRD | 2nd Grad |
| GTO | GTO |
| I EQ | INV 2nd x=t † |
| I GE | INV 2nd x≥t † |
| I Σ+ | INV 2nd Σ+ † |
| ICOS | INV 2nd cos † |
| IDMS | INV 2nd D.MS † |
| IDSZ | INV 2nd Dsz † |
| IFF | 2nd If flg |
| IFIX | INV 2nd Fix † |
| IIFF | INV 2nd If flg † |
| IINT | INV 2nd Int † |
| ILNX | INV ln x † |
| ILOG | INV 2nd log † |
| IND | 2nd Ind |
| INS | See Note Below |
| INT | 2nd Int |
| INV | INV |
| IPD* | INV 2nd Prd 2nd Ind † |
| IP/R | INV 2nd P→R † |
| IPRD | INV 2nd Prd † |
| ISBR | INV SBR † |
| ISIN | INV 2nd sin † |
| ISM* | INV SUM 2nd Ind † |
| ISTF | INV 2nd St flg † |
| ISUM | INV SUM † |
| ITAN | INV 2nd tan † |
| I$\bar{x}$ | INV 2nd $\bar{x}$ † |
| IXI | 2nd \|x\| |
| IY$^x$ | INV $y^x$ † |
| LBL | 2nd Lbl |
| LNX | ln x |
| LOG | 2nd log |
| LRN | See Note Below |
| LST | 2nd List |
| NOP | 2nd Nop |
| OP | 2nd Op |
| OP* | 2nd Op 2nd Ind |
| PAU | 2nd Pause |
| PD* | 2nd Prd 2nd Ind |
| PG* | 2nd Pgm 2nd Ind |
| PGM | 2nd Pgm |
| P/R | 2nd P→R |
| PRD | 2nd Prd |
| PRT | 2nd Prt |
| RAD | 2nd Rad |
| RC* | RCL 2nd Ind |
| RCL | RCL |
| R/S | R/S |
| RST | RST |
| RTN | INV SBR |
| SBR | SBR |
| SIN | 2nd sin |
| SM* | SUM 2nd Ind |
| SST | See Note Below |
| ST* | STO 2nd Ind |
| STF | 2nd St flg |
| STO | STO |
| SUM | SUM |
| TAN | 2nd tan |
| WRT | 2nd Write |
| X ≷ T | x:t |
| X$^2$ | $x^2$ |
| $\bar{x}$ | 2nd $\bar{x}$ |
| IXI | 2nd \|x\| |
| 1/X | 1/x |
| $\sqrt{x}$ | $\sqrt{x}$ |
| Y$^x$ | $y^x$ |

**SYMBOLS**

| |
|---|
| Σ+ |
| π |
| ) |
| ( |
| – |
| + |
| × |
| ÷ |
| = |
| . |
| +/– |

NOTE: This instruction is only seen when its key is encountered while listing a program. Because the key code cannot be placed in program memory by pressing the key, the key code can only be a remnant from the edit of some other instruction and should be corrected.

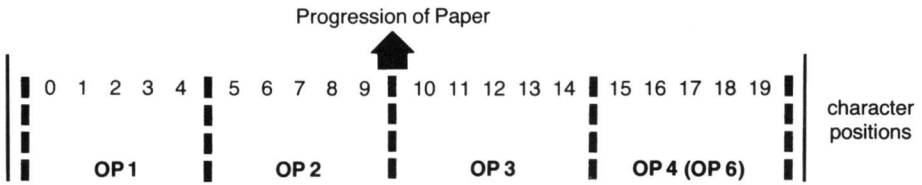
†Printed in trace mode only.

# SPECIAL CONTROL OPERATIONS FOR PRINTING

Special control operations 00 through 08 are specifically designed for use with the printer.

## Alphanumeric Printing — Op 00-06

The first seven control operations allow you to create and print out alphanumeric messages. Twenty characters can be printed on each line. They are assembled and stored in groups of 5 characters at a time as shown below.




Each printed character is represented by a two-digit, row-column address code according to the following table:

| TENS DIGIT | UNITS DIGIT 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | blank | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 7 | 8 | 9 | A | B | C | D | E |
| 2 | - | F | G | H | I | J | K | L |
| 3 | M | N | O | P | Q | R | S | T |
| 4 | . | U | V | W | X | Y | Z | + |
| 5 | × | ÷ | ( | ) | = | , | | |
| 6 | ↑ | % | ; | / | = | ' | × | Σ |
| 7 | ≥ | ? | ÷ | ◊ | Π | ∴ | Π | Σ |

For instance, A is code 13 and + is code 47. The codes for five characters (10 digits) can be entered into the display at a time. If you do not specify all 10 digits, zeros are assumed to precede the digits entered (each zero pair represents a blank space). To obtain spaces after some characters, simply enter pairs of zeros after the codes of the characters.

Once the display contains a series of character codes, press [2nd] [Op] **01, 02, 03** or **04** to tell the calculator exactly where on the line these characters are to be printed.

[2nd] [Op] **01** — for far left quarter of line
[2nd] [Op] **02** — for inside left quarter of line
[2nd] [Op] **03** — for inside right quarter of line
[2nd] [Op] **04** — for far right quarter of line

Pressing [2nd] [Op] **00** clears the print register. [2nd] [Op] **05** instructs the calculator to print the contents of the print register. Annotation can be used in the calculate mode to label segments of the paper tape or within a program for the same purpose.

Example: Title a paper tape "$\pi^2$ VS X% TESTS 3/22"

| Symbol | $\pi$ | $^2$ | | V | S | | X | % | | T | E | S | T | S | | 3 | / | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Code | 53 | 70 | 00 | 42 | 36 | 00 | 44 | 61 | 00 | 37 | 17 | 36 | 37 | 36 | 00 | 04 | 63 | 03 | 03 |

| Press | Display | Comments |
|---|---|---|
| [CLR] [2nd] [Op] **00** | 0. | Clear print register |
| 5370004236 [2nd] [Op] **01** | 5370004236. | Store "$\pi^2$ vs" printing on far left 1/4 |
| 44610037 [2nd] [Op] **02** | 44610037. | Store "X% T" for printing on inside left 1/4 |
| 1736373600 [2nd] [Op] **03** | 1736373600. | Store "ESTS" for printing on inside right 1/4 |
| 463030300 [2nd] [Op] **04** | 463030300. | Store "3/22" for printing on far right 1/4 |
| [2nd] [Op] **05** | 463030300. | Prints complete title on printer |

Note that a blank is the first thing needed for the inside left quarter. Nonreplaced leading zeros in the print register produce this blank. On the far right 04 is the first character code needed, but only the 4 need be entered. The quarters can be loaded in any order and can be written over by another set of annotation. The registers used for storing this annotation are normally used for the 5th, 6th, 7th and 8th pending operations. So do not attempt to have more than 4 pending operations when assembling annotation.

Be sure to remove all fix-decimal, scientific notation and engineering display formats before attempting to enter alphanumeric messages.

# VI

## Printer Control

A special-purpose control operation, 2nd Op **06,** prints the value currently in the display along with only the far right 4 characters on one line. This is especially useful to label program results.

Example: Design a program to calculate and label the pi approximation 22/7.

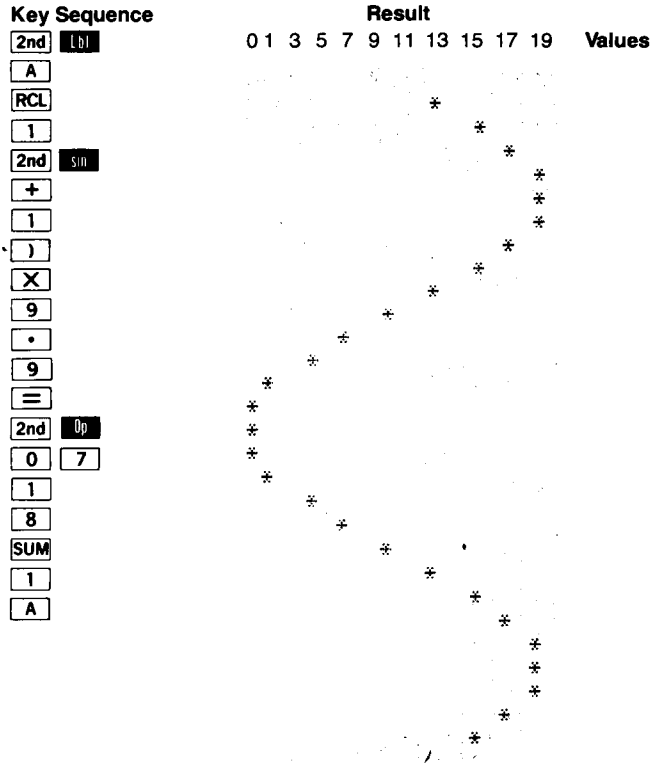
| Location and Key Code | Key Sequence | |
|---|---|---|
| 003 03 | 3 | P |
| 001 03 | 3 | P |
| 002 02 | 2 | I |
| 003 04 | 4 | I |
| 004 69 | 2nd Op | Store PI for printing |
| 005 04 | 0 4 | Store PI for printing |
| 006 02 | 2 | |
| 007 02 | 2 | |
| 008 55 | ÷ | |
| 009 07 | 7 | |
| 010 95 | = | 22 ÷ 7 |
| 011 69 | 2nd Op | Print |
| 012 06 | 6 | Print |
| 013 91 | R/S | |

When this program is executed, the printer will produce 3.142857143 Pi.

## Plotting Data — Op 07

Special control operation 07 plots a * for current display value (0-19) in character position 0-19 with 0 being on the left. Primarily designed for use in a program, this operation allows you to plot curves or histograms. Only one * is plotted per line and the integer part of the value X to be plotted must be $0 \leq X < 20$. If the displayed value is not within this range, the value is not plotted and the display flashes when the program halts. Only the integer portion is plotted.

Example: Design a program to plot a sine curve sampled every 18 degrees.




Notice that the sine of all angles is made positive by adding 1 to all sine values at location 004, 005. The values can now range from 0 to 2. If the values are scaled up by 9.9, the range is increased to 0 to 19.8, ideal for plotting purposes. The program is executed by storing a starting angle in register 01 then pressing [A]. The program continues until [R/S] is pressed.

## List Program Labels Used — Op 08

To obtain a sequential listing of all labels and the locations at which they occur in program memory, press 2nd Op **08.** This listing begins at the current position of the program pointer, so to list all the labels, press GTO **0** or RST before 2nd Op **08.** See the sample listing below.

| Program Location | Key Code | Audit Symbols |
|---|---|---|
| 001 | 11 | A |
| 018 | 12 | B |
| 062 | 19 | D' |
| 129 | 13 | C |
| 205 | 88 | DMS |
| 239 | 70 | RAD |
| 273 | 80 | GRD |
| 282 | 14 | D |
| 288 | 15 | E |
| 294 | 16 | A' |
| 300 | 10 | E' |
| 306 | 28 | LOG |

## PRINTER HEAD CLEANING PROCEDURE

The procedure for cleaning the print head is given in the *CARING FOR THE PRINTER* section of the PC-100A Owner's Manual. The following program should be used for the TI Programmable 58 and 59.

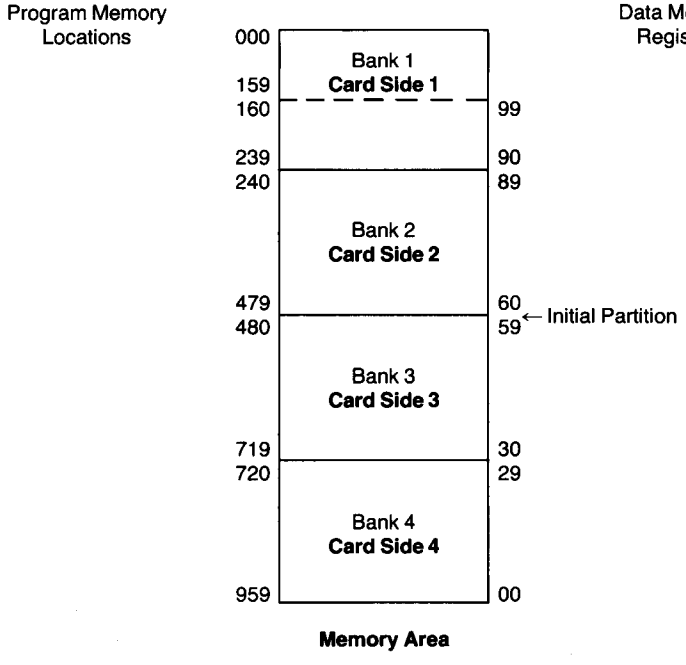
| Location and Key Code | Key Sequence |
|---|---|
| 000 04 | [4] |
| 001 42 | [STO] |
| 002 00 | [0] [0] |
| 003 09 | [9] |
| 004 42 | [STO] |
| 005 06 | [6] |
| 006 52 | [EE] |
| 007 01 | [1] |
| 008 00 | [0] |
| 009 94 | [+/-] |
| 010 22 | [INV] |
| 011 52 | [EE] |
| 012 35 | [1/x] |
| 013 76 | [2nd] [Lbl] |
| 014 11 | [A] |
| 015 84 | [2nd] [Op] [2nd] [Ind] |
| 016 00 | [0] |
| 017 97 | [2nd] [Dsz] |
| 018 00 | [0] |
| 019 11 | [A] |
| 020 76 | [2nd] [Lbl] |
| 021 12 | [B] |
| 022 69 | [2nd] [Op] |
| 023 05 | [5] |
| 024 97 | [2nd] [Dsz] |
| 025 06 | [6] |
| 026 12 | [B] |
| 027 91 | [R/S] |

To run this sequence, press [RST] [R/S]. Repeat if necessary.

# VII MAGNETIC CARDS

## TI PROGRAMMABLE 59 ONLY

You may permanently record any program and any data stored in the calculator on blank magnetic cards furnished with your calculator. As you should already know, your calculator is equipped with 120 storage registers which may be distributed between program memory and data memory. These registers are divided into four banks of thirty registers each. A card is designed to record two of these banks, one to a side.




**Memory Area**

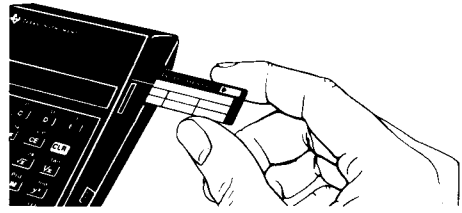
# Magnetic Cards

# VII

## TI PROGRAMMABLE 59 ONLY

## RECORDING CARDS

Magnetic cards are recorded using the [2nd] [Write] sequence. To record the contents of bank **n** (n = 1, 2, 3 or 4) onto card side **n**, press **n** [2nd] [Write] and insert the card (printed side up) into the lower slot in the right side of the calculator. Note that Fix 0 is the only fix-decimal option which will allow recording a card. If you are not sure of the display format, press [INV] [2nd] [Fix] before the recording key sequence.



The contents of that bank of registers, whether program information, data or both, are written onto the card. The card side you record always carries the bank number that the information is stored in. Only the integer portion of the displayed number is considered when [2nd] [Write] is pressed so fractional parts are ignored. For example, **2.31** [2nd] [Write] places the contents of bank 2 onto a card side that it labels 2. Any number of **n** whose magnitude is less than 1 or greater than or equal to 5 causes the display to flash and no recording takes place. So, be sure a 1, 2, 3 or 4 is in the display when you start to write a card.

When you insert a card into the calculator mechanism, do not restrict its advance once it is caught by the drive motor. The display remains blank until recording is completed, then the number of the recorded bank is displayed. If the bank number is flashing after writing, clear the display and perform the writing procedure again. If the display still flashes, your card may be faulty, try another card.

When recording data instead of a program, remember that data register 00 is at the end of bank 4 and the data registers number into bank 3, etc.

| 1 ◀ | TEXAS INSTRUMENTS | | | ▶ 2 |
|---|---|---|---|---|
| Commodity Spreads (Soybeans) | | | | 239.89 |
| | Limit | | | |
| Beans (¢) | Meal (¢) | Oil (¢) | Crush | Init. |

You should always label a magnetic card according to the information stored on it. In the upper corners of a card, spaces are provided to indicate the bank numbers recorded on that card. The arrow in each space shows which direction the card was entered into the calculator when recording the indicated bank. The space across the center of the card is available for a program title and other pertinent information such as the required partition. Below this line are two rows of boxes. The bottom five boxes may be used to indicate the function of the user-defined keys [A] through [E] within the recorded program. The upper row of boxes may be used similarly for the keys [2nd] [A'] through [2nd] [E'].

Note that when a register bank is recorded, the number of the bank and the current partitioning of the calculator are magnetically recorded on the card.

# VII

## Magnetic Cards

### TI PROGRAMMABLE 59 ONLY

**RECORDING MAGNETIC CARDS**

| Display When 2nd Write Pressed, Card Entered | Normal Program | Protected Program |
|---|---|---|
| **1, 2, 3, 4** | Writes a card side with this number from the bank of this number (program and/or data) and records current partition on card. | If bank contains only program, card is passed but not written — display number flashes.<br>If bank contains some data, bank is recorded but not protected with minus side number on the card. |
| **−1, −2, −3, −4** | Writes and protects card side with this number from the bank with this number. Also records current partition on card. | If bank contains only program, card is passed but not written — display number flashes.<br>If bank contains some data, bank is recorded with minus side number. |
| **Any other number** | Card is passed, but not recorded. Rightmost two integer digits of display are flashed. | Same as for normal program. |

If the display is flashing when trying to write a card, the card is passed but no recording takes place and the rightmost two integer digits are flashed.

The calculator should not be in fix-decimal format when recording cards.

For any flashing display the Drive motor continues to run until the card is removed.

# Magnetic Cards

# VII

## TI PROGRAMMABLE 59 ONLY

## PROTECTING A PROGRAM

If you have confidential information in a program to be written on a card, you can protect that information by entering a negative bank number for **n** before recording your card. Pressing **n** [+/−] [2nd] [Write] writes a card whose program can be read back into program memory and executed, but that's all. Restrictions on its use once it is read back into the calculator are listed below.

- Program cannot be listed or traced by a printer
- Program labels cannot be listed
- Cannot enter learn mode, edit, repartition or rerecord the program
- Cannot single-step execute or hold the pause key down during execution

You cannot force the calculator to read a protected card side into the wrong memory bank. When one side of a protected program card is read into program memory, an internal program protection flag is set, initiating the above restrictions. The only way to eliminate these restrictions (reset the flag) is to press [2nd] [CP] or turn the calculator OFF. You can read or download over a protected program in program memory, but this does not reset the protection flag. The "new" program now is protected as was the previous one.

You cannot protect the contents of the data registers. Also, a memory bank that is partitioned somewhere in the middle cannot be protected. Data registers are positioned at the front of a bank and the calculator sees the whole bank as a data storage area.

A nonprotected program can be written onto a card previously containing a protected program using the normal card writing procedures mentioned earlier. The "new" program on the card is not protected.

# VII

## Magnetic Cards

**TI PROGRAMMABLE 59 ONLY**

## READING CARDS

The drive motor of your calculator automatically pulls a magnetic card through the calculator when it is inserted into the card slot in the calculate mode. Whether or not the card is read depends upon the contents of the display register and the partitioning of the calculator as outlined below.

1. With zero in the display, any bank may be read providing the partitioning of the calculator and recorded program are the same. If the calculator is improperly partitioned, no read occurs, and the number of the bank recorded on the entered card is flashed in the display.
2. With **n** in the display the calculator may only read bank **n.** If any other bank number is on the card, the number read is flashed in the display and no read occurs. Again, if the partitioning is incorrect, the number of the entered bank is flashed in the display and the card is not read.
3. With **−n** in the display any bank read into the calculator is placed in bank **n** without question. Even partitioning is ignored. Once a card is read the bank number magnetically recorded on the card is placed in the display. A protected card (negative card number) cannot be forced into any bank other than the one numbered on the card.

If zero flashes in the display after a card is entered, the calculator has detected a misread and the card should be reentered.

If any number other than zero or ±**n** (n = 1, 2, 3 or 4) is in the display when a card is entered, no read occurs, and the right two digits of the display flash.

## READING A CARD FROM A PROGRAM

While running a program, [INV] [2nd] [Write] instructs the calculator to read a magnetic card in accordance with the above rules. A card can be inserted into the calculator card reader, but not read until the [INV] [2nd] [Write] instruction is encountered in the program. This allows data to be input where needed.

Remember that a card inserted into the card reader will automatically read when a program stops as a result of [INV] [SBR] or [R/S].

# Magnetic Cards

# VII

## TI PROGRAMMABLE 59 ONLY

**READING MAGNETIC CARDS**

| Display When Card Entered | Card with Normal Program | Card with Protected Program |
|---|---|---|
| **0** | Reads information into bank number listed on card if current partition matches that on card.<br>If partition incorrect, card is passed, but not read — display flashes card side passed. | Same as for normal program |
| **1, 2, 3, 4** | Expects card with this side number to be read — displays that side number.<br>If another side is entered or if partition is incorrect, card is passed but not read — display flashes card side passed. | If side passed numerically matches the display, card is read and side is displayed as negative.<br>If another side is entered or if the partition is incorrect, card is passed but not read — display flashes card side passed. |
| **−1, −2, −3, −4** | Forces side read into this bank number regardless of card number or partition.<br>A protected program cannot be forced into any bank. | Expects card with this side number to be read — displays that side number.<br>If another side is entered or if partition is incorrect, card is passed but not read — display flashes card side passed. |
| **Any other number** | Card is passed but not read — rightmost two integers in display flash. | Same as for normal program |

If the display is flashing any value when trying to read a card, the card is passed but not read and the rightmost two integers in the display are flashed.

The calculator should not be in fix-decimal format when reading cards.

Be sure your calculator battery pack is well charged or connected to an AC outlet before attempting any lengthy calculations. This is especially critical to the accuracy of reading and writing magnetic cards.

A flashing 0 in the display after reading a card indicates that the card has not been read correctly and should be reread.

# CARING FOR MAGNETIC CARDS

**CAUTION: Recorded magnetic cards may be damaged or altered if exposed to dust or foreign materials, permanent magnets, or electromagnetic fields (electric motors, power transformers, etc.).**

Magnetic cards have the ability to retain information placed on them for an indefinite period of time. The recorded information does not tend to fade or weaken with age and will remain unchanged until actually altered by an external magnetic field. While the magnetic signal will not deteriorate, the physical characteristics of the card and the card drive unit in the calculator are susceptible to damage.

## Handling Cards

Developing good habits in handling magnetic cards is important. A card which is physically marred, creased or dented may be useless for its intended purpose. However, physical degradation of a card generally results from an accumulation of mishaps or poor handling techniques.

There are numerous contaminants to consider. Ashes, food particles, drinks, dust and oil-based liquids are the most common contaminants to guard against. A card can be contaminated by placing it directly on a contaminated surface; or indirectly, by transferring the contaminants to the card with your fingers. Even the natural oils on your fingers will transfer to the cards and cause accumulation of dust and foreign particles. Note that using one contaminated card in the calculator may contaminate not only the calculator card reading mechanism, but also other cards which are used later. In some cases of extreme contamination by oily materials, the calculator card reading mechanism can be rendered inoperative and require repairs by a Texas Instruments Service Facility. The following simple instructions are important to assure maximum life of the magnetic cards.

1. Handle a card by its edges whenever possible.
2. Keep the cards away from magnets and sharp objects that could scratch the oxide coating.
3. Keep the card in the vinyl carrying case or other protective container while the card is not in use.
4. If a card is contaminated, clean it immediately.
5. Do not attempt to read or write visibly damaged or contaminated cards.

# Magnetic Cards

# VII

**TI PROGRAMMABLE 59 ONLY**

## Cleaning Cards

Contaminated card may be cleaned easily without using special cleaners or solvents. Petroleum based fluids such as lighter fluid should not be used to clean cards under any circumstances. Dust and foreign particles should be removed from a card with a soft dry cloth. Other forms of contamination may be washed from the card with warm water and a small amount of mild liquid detergent. Rinse the card and dry with a soft cloth.

## Marking On Cards

The blank magnetic cards furnished with your calculator have areas designated for you to write numbers, symbols and abbreviated titles for your personal programs. You may write information temporarily on a card with a soft fine-lead pencil or a fine-point, felt-tip pen with washable ink. Of course, a felt-tip pen with non-washable or permanent ink will permanently mark your card. For best results, check with your local school supply outlet and ask for felt-tip pens that are used to write on transparencies. Most outlets carry a variety of colors with washable or permanent inks.

## USING THE HEAD-CLEANING CARD

The specially marked head-cleaning card furnished with your calculator has an abrasive coating in place of the usual oxide. Using this card will remove any build-up of oxide or foreign particles from the magnetic read/write head in the calculator. This card should not be used as an all-purpose remedy for any difficulty experienced, as excessive use could change the characteristics of the read/write head. The *IN CASE OF DIFFICULTY* of *Appendix A* gives instructions that should be used as the guide for when the head cleaning card may be used to remedy a difficulty. To use the card; insert the card into the lower slot of the calculator as you would a regular card, and let the drive motor pull the card through the calculator. Press [CLR] if the display flashes after using the card. The head cleaning card should be used sparingly and no more than one time per difficulty. Be sure this card is clean before using it.

## USING THE DRIVE ROLLER CLEANING CARD

The Drive Roller Cleaning Card is to be used about every 500 reads or when a card begins to slip or move at a nonuniform pace through the calculator. Press [1] [2nd] [Write] and insert the card. Hold onto the tail-end of the card moving it back and forth while the drive mechanism is in motion. Three or four seconds of this action should be sufficient to properly clean the roller. Withdraw the card and press [R/S]. If the drive motor continues running after removing the card, it may be necessary to turn the calculator off and on again.

# USING THE CALCULATOR DIAGNOSTIC CARD

The CALCULATOR DIAGNOSTIC magnetic card may be used to check the functional status of the calculator and its magnetic card read/write operation. This diagnostic does not interact with or check the operation of a library module.

## Calculator Test

To use the CALCULATOR DIAGNOSTIC card, read side 1 of the card by pressing [CLR] and inserting the card into the lower slot of the calculator. After the card is read, the display should show 1. Press [E] to perform the diagnostic. A display of −.8888888888 indicates the calculator passed the test. When run using the PC-100A Printer, the displayed value is also printed. A flashing display indicates the calculator failed the test. Repeat test to verify that problem is real. If so, perform the following Read/Write Test to see if the card reader is at fault.

## Read/Write Test

This test may be more convenient to perform with the PC-100A Printer. Read side 2 of the CALCULATOR DIAGNOSTIC card by pressing [CLR] and inserting the card into the lower slot of the calculator. After the card is read, the display should show 1. If the Printer is used, press [RST] [2nd] [List] and verify that the calculator contains key code 77 in locations 000 through 229. This may be done manually by pressing [RST] [LRN] and [SST] to verify that 77 is in each location. Other than 77 at any location from 000 through 229 indicates a card reading problem. Repeat complete test procedure to be sure the problem is real.

To check the card-write function of the calculator, read side 2 of the CALCULATOR DIAGNOSTIC card as described above. Now use a blank magnetic card, press [CLR] [1] [2nd] [Write] and insert blank card into the lower slot of the calculator. Press [2nd] [CP] and read the card just recorded into the calculator. Check that 77 is in locations 000 through 229. Repeat complete test if an error is found to be sure the problem is real.

# A

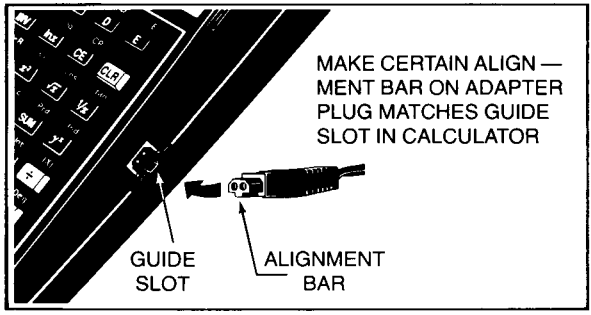
## Appendix

### MAINTENANCE AND SERVICE INFORMATION

## BATTERY AND AC OPERATION

**Normal Operations** — To ensure maximum portable operating time, connect the **AC9900H-UK** Adapter/Charger to a standard **240V/50Hz** outlet, plug into calculator, and charge battery pack at least 4 hours with the calculator OFF or 10 hours with the calculator ON. The adapter/charger and battery pack may become warm when used on AC power. This is normal and of no consequence.

**CAUTION: The calculator can be damaged if the adapter/charger is connected without the battery pack installed.**




When the battery pack is fully charged, the calculator will operate approximately 2 to 3 hours before recharging is necessary. However, don't hesitate to connect the adapter/charger if you know or suspect the battery pack is nearly discharged. A battery pack near discharge can adversely affect all calculator operations. A discharged battery pack is typically indicated by a dim, erratic or blank display, or the card drive motor starts to run. If a battery pack becomes completely discharged while reading a card, the program on the card can be erased or altered.

While individual cell life in a battery pack is difficult to predict, under normal use, rechargeable batteries have a life of 2 to 3 years or about 500 to 1000 recharge cycles.

**Periodic Recharging** — Although the calculator will operate indefinitely with the adapter/charger connected, the rechargeable battery pack can lose its storage capacity if it is not allowed to discharge occasionally. For maximum battery life, it is recommended that you operate the calculator as a portable at least twice a month, allowing the batteries to mostly discharge, then recharge accordingly.

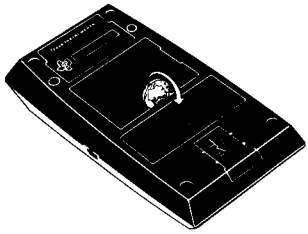
**Excessive Battery Discharging** — If the calculator is left on for an extended period of time after the battery pack is discharged (accidentally left on overnight, for example), connect the adapter/charger for at least 24 hours with the calculator OFF. If this does not restore normal battery operation the battery pack should be replaced. Repeated occurrences of excessive battery discharging will permanently damage the battery pack. Spare and replacement BP-1A battery packs can be purchased from your local TI **retailer or directly from Texas Instruments Ltd, E.C.D., Block C, Manton Centre, Manton Lane, Bedford. For other countries use local service center address.**

**Storage** — If the calculator is stored or unused for several weeks the battery pack will probably need recharging before portable use. The battery pack will not leak corrosive material; therefore, it is safe to store the calculator with the battery pack installed.

# Appendix A

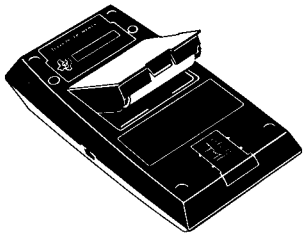
## MAINTENANCE AND SERVICE INFORMATION

**Battery Pack Replacement** — The battery pack can be quickly and simply removed from the calculator. Hold the calculator with the keys facing down. Place a small coin (penny, dime) in the slot on the back of the caculator. A slight prying motion with the coin will pop the slotted end of the pack out of the calculator. The pack can then be removed entirely from the calculator.



The exposed metal contacts on the battery pack are the battery terminals. Care should always be taken to prevent any metal from coming into contact with the terminals thereby shorting the batteries. These terminals should be cleaned periodically with a pencil eraser to remove any corrosion that may have accumulated.

To reinsert the battery pack, place the rounded part of the pack into the pack opening so that the small step on the end of the pack fits under the edge of the calculator bottom. The slotted end of the pack will then be next to the caution instruction. A small amount of pressure on the battery pack will snap it properly into position.



# A

Appendix

## MAINTENANCE AND SERVICE INFORMATION

## IN CASE OF DIFFICULTY

In the event that you have difficulty with your calculator the following instructions will help you to analyze the problem. You may be able to correct your calculator problem without returning it to a service facility. If the suggested remedies are not successful, contact the Consumer Relations Department by mail or telephone (refer to *If You Have Questions or Need Assistance* later in this appendix). Please describe in detail the symptoms of your calculator.

If one of the following symptoms appears while operating with the optional printing unit, remove the calculator and reinstall the battery pack. If the symptom disappears when the calculator is removed from the printing unit, refer to the printing unit manual.

| Symptom | Remedy |
|---|---|
| 1. Display shows erroneous results, flashes erratic numbers, grows dim, goes blank, or the card reader runs continuously. | The battery pack is probably discharged. Refer to *Battery and AC Operation* at the first of this appendix. |
| 2. Display is blank for no obvious reason. | Press and hold [R/S] momentarily. If display returns, the calculator was running a long program, hung in a loop, or waiting for a card to be inserted.<br><br>Press and hold [RST] momentarily. If display returns, processing was in the library module, either hung in a loop (possibly because of a low battery) or was running a long program.<br><br>The battery pack may be discharged or improperly installed. |
| 3. Display flashes while performing keyboard operations. | An invalid operation or key sequence has been pressed or the limits of the calculator have been violated. See Appendix B for a list of these conditions. |
| 4. Display flashes each time a library program is called. | Library program number does not exist. Consult library manual.<br><br>Library module not properly installed. Refer to Section III of this manual. |

# Appendix A

## MAINTENANCE AND SERVICE INFORMATION

| Symptom | Remedy |
|---|---|
| 5. Display flashes or produces incorrect results when running a library program. | The wrong program may have been called.<br><br>Improper operating procedure. Check related User Instructions in the library manual.<br><br>Partitioning is set for too few data registers to run the program.<br><br>Calculator is operating in Fix-Decimal display format. Press [INV] [2nd] [Fix] or turn calculator OFF and ON and try program again.<br><br>Press [CLR] [2nd] [Pgm] 1 [SBR] [=] to run the library diagnostic. If the result is flashing, check that library module is properly installed (see Section III) and press this sequence again. |
| 6. Display flashes or produces incorrect results when running a personalized program in program memory. | An illegal operation, overflow or underflow occurred while the program was running. Appendices B, C and D may be useful in finding the problem.<br><br>One of the library programs has been called. Press [RST] and try again.<br><br>Press [CLR] [2nd] [Pgm] 1 [SBR] [=] to perform the library diagnostic. If the result is flashing, review actions in symptom 5.<br><br>If program has been read from a magnetic card, perform the tests given in *Using the Calculator Diagnostic Card* at the end of Section VII. If the test results are good, check the magnetic card with your program for physical defects or contamination. Review actions in symptom 7. |

| Symptom | Remedy |
|---|---|
| 7. Display flashes after reading or recording a magnetic card. | Improper procedure. Refer to Section VII.<br><br>Incorrect partitioning selected.<br><br>A reading error has been detected. If other cards read properly, check first card for physical defects or contamination, and clean or replace card as necessary. If the card was contaminated, see *Using the Driver Roller Cleaning Card* in Section VII. If other cards do not read properly, use the head cleaning card one time — refer to *Using the Head Cleaning Card, Using the Drive Roller Cleaning Card,* and *Using the Calculator Diagnostic Card* in Section VII. |
| 8. Calculator will NOT go into learn mode, single step, list or record a card. | The program in the program memory is protected. See *Protecting a Program in Section VII.* |

When returning your calculator for repair, return the calculator, adapter/charger, library module and any magnetic cards which were involved when the difficulty occurred. For your protection, the calculator must be sent insured; Texas Instruments cannot assume any responsibility for loss of or damage to uninsured shipments.

# Appendix A

## MAINTENANCE AND SERVICE INFORMATION

If the calculator is out of warranty, service rates in effect at time of return will be charged. Please include information on the difficulty experienced with the calculator as well as return address information including name, address, city, state and zip code. The shipment should be carefully packaged, adequately protected against shock and rough handling and sent to one of the Texas Instruments Service Facilities listed with the warranty.

## IF YOU HAVE QUESTIONS OR NEED ASSISTANCE

If you have questions concerning calculator repair, accessory purchase or the basic functions of your calculator, please telephone or write to the Consumer Relations Department of the nearest Texas Instruments facility listed on the back cover of this book. Technical questions such as programming, specific calculator applications, etc., should be asked of the Technical Department at the same address.

Because of the number of suggestions which come to Texas Instruments from many sources containing both new and old ideas, Texas Instruments will consider such suggestions only if they are freely given to Texas Instruments. It is the policy of Texas Instruments to refuse to receive any suggestions in confidence. Therefore, if you wish to share your suggestions with Texas Instruments, or if you wish us to review any calculator program key sequence which you have developed, please include the following statement in your letter:

> "All of the information forwarded herewith is presented to Texas Instruments on a nonconfidential, nonobligatory basis; no relationship, confidential or otherwise, expressed or implied, is established with Texas Instruments by this presentation. Texas Instruments may use, copyright, distribute, publish, reproduce, or dispose of the information in any way without compensation to me."

# B

## Appendix

## ERROR CONDITIONS

A flashing display indicates that the displayable limits of the calculator have been violated or that an invalid calculator operation has been requested. Pressing [CE] or [CLR] stops the flashing. [CLR] also clears the display and pending operations. [CE] stops the flashing only, permitting further calculations with undisturbed pending operations. The display flashes for the following reasons:

1. Calculation entry or result (in display or memories) outside the range of the calculator, $\pm 1 \times 10^{-99}$ to $\pm 9.9999999 \times 10^{99}$. The exceeded limit is flashed, indicating underflow or overflow.

2. Inverse of a trigonometric function with an invalid value for the function such as $\sin^{-1} x$ with x greater than 1. The invalid value x is flashed.

3. Root or logarithm of a negative number. The root or logarithm of the absolute value of the number is flashed to indicate the sign error.

4. Raising a negative number to any power (or root). The power (or root) of the absolute value of the number is flashed.

5. Pressing two operation keys in succession. This affects $+$, $-$, $\times$, $\div$, $y^x$, and $\sqrt[x]{y}$. The last entered number is flashed.

6. Pressing [=] or [)] after $+$, $-$, $\times$, $\div$, $y^x$, or $\sqrt[x]{y}$ or following a number with [(] with no intervening operation. The last entered number is flashed.

7. Having more than 9 open parentheses or more than 8 pending operations. The 10th parenthesis or 9th operation is not accepted so processing can continue. The last displayed number is flashed.

8. Dividing a number by zero. "9.9999999 99" is flashed.

9. Calling for a special control operation outside the range 00-39.

10. Attempting to plot ( [0] **07**) outside the range 00-19. The display value flashes.

11. Attempting to partition beyond the limits of the TI Programmable 58 (60 data registers, 00-59). The TI Programmable 59 uses 100 data registers if more than 10 sets are requested.

12. Attempting to transfer to unassigned label positions or outside of the program partition to nonexistent program locations causes the display to flash.

13. Attempting to download a library program where insufficient program memory space exists causes the display to flash.

14. Addressing a nonexistent library program number flashes the current display value.

15. In linear regression calculations, if the line parallels the y-axis, attempting to calculate slope, intercept, correlation, x′ or y′ will cause flashing. If the line parallels the x-axis, the display flashes when attempting to calculate x′ or correlation.

# Appendix B

## ERROR CONDITIONS

16. Calculation of slope, intercept, correlation, x′ or y′ with fewer than 2 data points entered. The last displayed number flashes in the display.

17. Having more than four pending operations during linear regression, trend-line analysis or statistical routines, or during polar/rectangular or degrees-minutes-seconds conversions.

18. $0^{-x}$ and $\sqrt[x]{0}$ produces flashing overflow "9.9999999 99".

19. Radius outside the range $10^{\pm 50}$ in Rectangular to Polar Conversions. The angle is flashed.

20. Arguments that do not satisfy the following limits cause a flashing display.

| **Function** | **Limit** |
|---|---|
| $\sin^{-1}x$, $\cos^{-1}x$ | $-1 \leq x \leq 1$ |
| $\ln x$ $\log x$ | $1 \times 10^{-99} \leq x < 1 \times 10^{100}$ |
| $e^x$ | $-227.9559242 \leq x \leq 230.2585092$ |
| $10^x$ | $-99 \leq x < 100$ |

21. Odd multiples of $\pm$ 90°, $\pm \pi/2$ radians, and $\pm$ 100 grads are undefined points of the tangent function. Small multiples result in an overflow condition; however, multiples of this function yield incorrect results without giving an error indication. See **Appendix C** for more details.

22. Incorrect reading or writing of cards, flashes the display.

### ERRORS ENCOUNTERED WHEN RUNNING A PROGRAM

When any of the foregoing errors occur in a program, what happens next depends upon the programmer. Program halts are not an automatic consequence of an error condition except for number 12 above. The program continues, using the value that would have flashed in keyboard operation for subsequent calculations, and the presence of an error will be signaled by flashing the "answer" obtained when the program halts. This may or may not be the correct answer, depending upon the problem and the type of error condition. However, it is the best selection which can be made without further instructions from the programmer. If the programmer desires he may instruct the calculator to cease execution when an error condition arises by setting flag 8 or by using the error tests, Op 18 and 19.

# C

## Appendix

## DISPLAYED RESULTS VERSUS ACCURACY

Calculators, like all other electrical and mechanical devices, must operate with a fixed set of rules within preset limits.

The basic mathematical tolerance of the calculator is controlled by the number of digits it uses for calculations. The calculator appears to use 10 digits as shown by the display, but actually uses 13 digits to perform all calculations. Combined with the built-in 5/4 rounding capability, these extra digits guard the ten-digit display to improve accuracy. Consider the following example in the absence of these guard digits.

$$1/3 \times 3 = .9999999999 \text{ (inaccurate)}$$

The example shows that $1 \div 3 = .3333333333$ when multiplied by 3 produces an inaccurate answer. However, a thirteen-digit string of nines will *round* to 1 when rounded to 10 places.

The higher order mathematical functions use iterative calculations. The cumulative rounding error is usually maintained below the ten-digit display so that no effect can be seen. The 13-digit representation of a number is three orders of magnitude from the displayed tenth digit. In this way, the display assures that results are rounded accurately to ten digits.

Normally, there is no need to even consider these guard digits. On certain calculations, however, the guard digits may appear as an answer when not expected. The mathematical limits of finite operation: word length, truncation and rounding errors do not allow the guard digits to always be completely accurate. Therefore, when subtracting two functions which are mathematically equal, the calculator may display a nonzero result.

Example: $\sin 45° - \cos 45° \neq 0$

Select degree mode

| Press | Display |
|---|---|
| 45 [2nd] [sin] [−] | .7071067812 |
| 45 [2nd] [cos] | .7071067812 |
| [=] | 7.−13 |

The identical display results of sin 45° and cos 45° show that the functions are accurate to at least ten digits. The final result indicates a discrepancy in the thirteenth digit. The significance is that results smaller than entry or intermediate results off by a factor of $10^{-11}$ to $10^{-12}$ are potentially equal to zero.

The above fact is especially important when writing you own programs. When testing a calculated result to be equal to another value, such as [x=t], precautions should be taken to prevent improper evaluation due to the guard digit differences. The sequence [EE] [INV] [EE] will truncate the guard digits of a result leaving only the rounded display value for further use.

# Appendix C

## DISPLAYED RESULTS VERSUS ACCURACY

For the standard display, results are accurate for all calculations that do not violate the restrictions listed in *Appendix B*, except as defined below.

**TRIGONOMETRIC FUNCTIONS** — **All displayed digits in standard display format are accurate to ± 1 in the 10th digit** for a ± 36,000 degree range, ± 200 $\pi$ radians and ± 40,000 grads. When the argument range reaches ± $3.6 \times 10^{14}$ degrees (± $6.2799993 \times 10^{12}$ radians or ± $4.0 \times 10^{14}$ grads) or more, no partial rotation is recognized. In general, the accuracy decreases one digit for each decade outside the specified accuracy range. An exception is the tangent of an odd multiple of ± 90°, ± $\pi/2$ radians or ± 100 grads that results in an overflow condition because the function is undefined at these points.

**ROOTS AND POWERS** — There can be some accuracy loss for roots and powers in calculations when the base y gets very close to 1 *and* the power x gets very large. For example, $.99999944^{-160000}$ is accurate through 8 digits, whereas $.99999944^{-400}$ is accurate throughout all 10 standard display digits.

# D

## Appendix

## TROUBLESHOOTING PROGRAMS

Even the most cautious programmer occasionally runs into a situation where things just aren't working properly. Your calculator provides a set of keys to make editing easy. The problem is to find the errors. Naturally, you should first check to see that the program has been keyed in correctly. If no inconsistencies are found then the remainder of this Appendix can probably help you. A list of common programming errors is given first. Once you have familiarized yourself with these suggestions, you may recognize your mistake. If not, continue on into *Program Diagnosis,* the next part of this section.

The optional PC-100A printer becomes especially useful here for listing your program or tracing calculations.

# BASIC CONSIDERATIONS

## Algebraic Operating System

Most problems can be keyed into the calculator just as they are written, but this does not mean that they are interpreted in that same order. The algebraic hierarchy of the calculator causes an expression such as $2 + 3 \times 6$ to be interpreted as $(3 \times 6) + 2 = 20$, not $(2 + 3) \times 6 = 30$.

Another point to remember is that single-variable functions must *follow* the number being operated on. Sin $\pi$, for example, is evaluated by the sequence [2nd] [$\pi$] [2nd] [SIN].

## Equals Command — [=]

The equals instruction completes *all* pending operations; hence it should be used with discretion, especially in subroutines.

The *lack* of an equals can also cause problems. Consider the expression $(2 \times 3)^2$. The sequence [2] [×] [3] [$x^2$] causes the calculator to determine the value $2 \times 3^2$. The sequence required here is either [2] [×] [3] [=] [$x^2$] or [(] [2] [×] [3] [)] [$x^2$].

## Pending Operations

Your calculator can retain up to nine levels of parentheses and eight pending operations. However, some of the calculator's keyboard functions require the use of as many as four pending operations. Polar/rectangular and degree format conversions are among these functions, as well as the statistical functions. An entry that exceeds the limits is simply ignored and the display flashes to let you know what has happened.

## Multiple Labels

Each label may be used only once within a program. The label search mechanism always begins at location 000, not at the point where the label is called. Therefore, transfer always goes to the first label and a second use of that label could never be found.

# Appendix D

## TROUBLESHOOTING PROGRAMS

### Subroutine

Six subroutine levels are probably more than you will ever need. Calling a subroutine beyond the sixth level does *not* store a new return address in the subroutine return register. When [INV] [SBR] is encountered in a seventh level subroutine, processing returns according to the last or sixth address stored in the return register, which is, of course, not the place you expected. This condition, however, does not cause an error indication, so you may not be aware that an incorrect transfer has been made.

Keep in mind that a user-defined key is a subroutine call unless it is preceded by a transfer instruction such as [GTO] or [2nd] [$x \geq t$].

Also, polar/rectangular and degree format conversions and statistics each require one level of subroutine.

### Reset Instruction — [RST]

This instruction is very useful, but keep in mind all of the things it can do so you can avoid unintentional effects. [RST] performs four functions: positions the program pointer to location 000, resets all program flags, clears the subroutine return register and halts library programs, returning the pointer to program memory.

### Statistical Functions

When using the preprogrammed statistical functions, data is *summed* into the contents of data registers $R_1$-$R_6$. Therefore, a program using these functions should not only avoid the use of these registers, but should clear them as well before starting data entry. Steps should also be taken to preserve the value stored in the T-register if it is needed later in the program. As mentioned above, these functions require up to four pending operations and one subroutine level.

### Polar/Rectangular Conversions

Here the primary thing to remember is to select the correct angular mode. Again, these conversions use up to four pending operations and one subroutine level.

### Angular Mode Selection

Your calculator powers-up in the degree mode. If you want angles to be interpreted in either radians or grads you need to instruct the calculator to do so with the appropriate keys. The calculator then remains in the selected angular mode until another is chosen. There is no visible indication of which angular mode the calculator is in.

### Functions Operating on the Display Only

[EE] and [2nd] [D.MS] operate only upon the contents of the display, not the display register. That is, any guard digits and any digits suppressed by placing the calculator in a fix-decimal display format are lost when you use these instructions.

# D

Appendix

## TROUBLESHOOTING PROGRAMS

## T-Register Comparisons

[2nd] [x=t] and [2nd] [x≥t] compare the entire display register against the entire T-register when deciding whether or not to branch to a new location. As an illustration of the type of problem you may run into using these instructions, try the following sequence.

[2nd] [Deg]
45 [2nd] [sin]
[x:t]
45 [2nd] [cos]
[2nd] [x=t] 114

Now, if you enter the learn mode, you find that this transfer didn't take place even though sin 45° and cos 45° are mathematically equal. This is due to rounding that occurs in the guard digits when your calculator computes the values (see *Appendix C*). To prove this to yourself, subtract cos 45° from sin 45°. You get a nonzero result, indicating that these values differed in the guard digits. Normally, you would never detect this difference, but you need to keep it in mind when working with conditional transfers. The sequence [EE] [INV] [EE] will truncate the guard digits of a result, leaving only the rounded display value for further use.

Also, you should be careful when using the [EE] and [2nd] [D.MS] functions where guard digits are discarded or altered.

## Editing

You should use care when editing programs. Even simple changes, which may seem almost insignificant at one point, may cause complications elsewhere. Consider all possible effects that any change can make. Some things to watch out for are merged addresses, duplicate labels, merged instructions (like [INV] [SBR] ), and addresses that can be interpreted as key codes. You cannot edit a protected program.

Remember that adding and deleting instructions invariably moves parts of the program up and down. Transfer instructions using absolute addressing should be corrected accordingly.

## Partitioning

Be sure that the data registers and program memory you use are within your partition. A data register can hold 8 program instructions. Each pair of numbers in a data register is a potential instruction. So, be careful when repartitioning that the contents of a data register do not become 8 program instructions and vice versa.

# Appendix D

## TROUBLESHOOTING PROGRAMS

## PROGRAM DIAGNOSIS

Keeping the above in mind, here are some suggestions for program diagnosis. The intention here is to suggest ways for you to evaluate programs that do not function properly.

### Program Does Not Terminate

When a program does not terminate when expected, it is usually said to be "hung" in a loop. The best procedure is to step through the program analyzing each instruction and giving special attention to transfer statements. Though a branching instruction is likely to be the culprit, unconditional transfers should be checked first as an error here is easier to detect. Be wary of sequences such as [2nd] [Lbl] [D] . . . [GTO] [D] . Here a conditional transfer is required to provide a way out of the loop. Examine conditional transfers next, especially if they are designed to begin or end a loop.

The [2nd] [Dsz] instruction shouldn't send a program into an infinite loop unless the program does something to alter the contents of the data register being decremented or the register value is $\geq 10^{10}$. Make sure that you allow this data register to go to zero. If a very large number is stored in the data register that is being decremented, the program may take an exceptionally long time to terminate and only appear to be hung in a loop. However, if the program is designed to determine the number of loops that are needed, you may want to check this calculation.

Conditional transfers making T-register comparisons should also be carefully examined. Usually, when you use a transfer of this type to control a loop, you expect your calculations to converge within predetermined limits. Naturally, if your calculations don't converge, the loop doesn't terminate. Check both the equations you are using and the instructions you are using to program these equations. Also, make sure that the correct test value is stored in the T-register. If the program is designed to determine this number, you should examine these calculations as well. One additional problem (discussed previously in *T-Register Comparisons* under *CONDITIONAL TRANSFERS* in Section V) is that these instructions compare the *entire* T-register against the *entire* display register before deciding whether or not to branch to a new location.

If you are using a library program as a subroutine, you can press [RST] . If the program stops then the library program was hung in a loop. In this event, see the user instructions of the library routine you used. Pressing [RST] is an emergency measure and should be used only as a last resort, because the stopping point cannot be predicted and the displayed value cannot be identified.

If you unable to detect any error after completing this evaluation, refer to *Using the Calculator in Diagnosis* later in this section.

# D

Appendix



## Consistent Data Yields Inconsistent Results

Errors of this type are usually caused by conditional transfers that, when used improperly, can give correct results one time and not the next. As an illustration, consider the following example:

[2nd] [Lbl]
[A]
[INV]
[x≥t]
[B]
[2nd] [St flg]
[3]
[2nd] [Lbl]
[B]
•
•
•
[2nd] [If flg]
[3]
[C]
•
•
•
[2nd] [Lbl]
[C]
•
•
•

Here, the programmer wants to skip a portion of the program if his original data was less than zero. Let's assume that the following set of data is entered into the program: 12, −16, 12. The program produces correct results when the first two entries are made; however, when 12 is entered a second time, the program yields an incorrect result. This is because entering −16 caused flag 3 to be set. Then, since the program includes no provision for resetting flag 3 when positive entries are made, 12 is treated as a negative entry when it is entered a second time. This situation may be remedied by placing the sequence [INV] [2nd] [St flg] [3] following [2nd] [Lbl] [A].

Similar problems may occur with any transfer operation. Unfortunately, it isn't possible to give an example of each. As a general rule, however, the first thing to do when diagnosing a problem of this nature is to look for a pattern in the answers. In the above, for example, only positive entries produced wrong answers, and then only after a negative entry was made.

## TROUBLESHOOTING PROGRAMS

Of course, not all errors of this type are caused by transfer operations. Consider a situation in which consistently different answers (e.g., increasing according to a recognizable pattern) result from entering the same data several times in succession.

```
2nd Lbl
A
.
.
.
SBR
SUM
.
.
.
2nd Lbl
SUM
SUM
1 2
.
.
.
INV SBR
```

Here, the problem is caused by the careless use of a data register. If $R_{12}$ is never cleared, and unless the initial operation on this data register is a STO instruction, continuously increasing answers result.

As mentioned above, the key to this type of problem is to find a pattern in the answers. If you can't find a pattern without a lot of extra effort, see *Using the Calculator in Diagnosis*.

In most cases, such situations may be avoided by incorporating an initialization sequence into each of your programs. A typical initialization procedure is shown below. It is called by simply pressing [E].

| Key Sequence | Comments |
|---|---|
| [R/S] | Ends sequence (location 000) |
| [2nd] [Lbl] [E] | |
| [2nd] [CMs] | Clears data memories |
| [CLR] | Clears display |
| [2nd] [CP] | Zeros T-register |
| [INV] [2nd] [Fix] | Removes fix-decimal |
| [RST] | Resets all flags |
| | Clears subroutine register |
| | Sends program pointer to 000 |

## Consistently Wrong Answers

It is possible that a program in which the same erroneous answer occurs consistently, regardless of what data you enter, has been written using an incorrect solution. However, if you have manually worked through your equations, found them valid for all cases, and can find no instruction error, then you should refer to the following section.

## Using the Calculator in Diagnosis

Once you have determined what values should be computed and displayed, and where they should be stored at different stages of the program, your calculator can be used as the most efficient means of examining a malfunctioning program.

A strong word of caution: when you recall data register conents for examination at an inspection point, be sure that you restore the contents of the display register before continuing program execution. Otherwise, nonexistent program errors may appear to occur. It is a good idea to use [2nd] [Exc] to call the data to the display for examination and then replace it using the same sequences. This returns the most recently calculated value to the display.

There are several instructions that you may use to analyze a program as it is running. Inserting [R/S] commands at various key points in a program is a quick method of finding where an error first appears. When the program stops, you should check not only the contents of the dsiplay, but those of the data and test registers as well.

When inserting [R/S] instructions, it is easiest to start at the highest numbered program location end and work backwards. Inserting the Run/Stop in reverse order does not move around program locations yet to receive the instructions.

# Appendix D

## TROUBLESHOOTING PROGRAMS

Once a discrepancy is found, run the program again and stop at the [R/S] instruction ahead of the [R/S] where the error was detected. Now use the [SST] key to execute the program one step at a time until you find the exact location of the error. After the problem has been identified and the error corrected, delete the [R/S] instructions in the order they occur in program memory. Repeat this process until all errors have been found and corrected.

In place of using [SST], you can hold down the [GTO] key. This inserts a pause between each step as it's executed, allowing you a brief but automatic look at the progress of the program. To use this option you must first start the program. To make sure that you observe the results of all program instructions, beginning with the first one, follow these three steps.

1. Press and hold down the [R/S] key.
2. While holding down the [R/S] key, press and hold the [GTO] key.
3. Release the [R/S] key.

The most easily diagnosed problem is one that results in an error condition. When this occurs, simply set flag 8 and run the program again. Now, when the error is encountered, the program halts. Pressing [LRN] shows you the program location where the error occurred. (Processing actually stops with the program pointer on the first location following the error.) You should then be able to determine the nature of the error and make necessary correction.

Note: You may not single-step through a library program. When single-stepping through program memory that calls a library program, the display goes virtually blank while the library program is being executed. Once this routine is complete, you may continue to observe the main program using either [SST] or [GTO]. (If you are using [GTO], do not release this key while a library routine is being executed or part of the main routine may escape inspection.)

## Using the PC-100A Printer in Diagnosis

The optional PC-100A is another valuable aid in diagnosing a program. With your calculator on the print cradle, here's what you can do.

(1) Press [RST] [2nd] [List] to obtain a complete listing of your program instructions including program location numbers, instruction codes, and instruction mnemonics. This saves you the trouble of single-stepping through a program in the learn mode and translating instruction codes into program instructions to verify that you've entered your program correctly.

(2) Run a program with the printing unit in the trace mode. This enables you to easily follow the sequence of calculations step-by-step and discover exactly where the program deviates from the solution that you intended.

(3) Press **XX** [INV] [2nd] [List] to obtain a list of data register contents beginning with $R_{xx}$. By stopping a program at various key points and performing this operation, you can easily verify that the data registers contain the right quantities at the right times.

(4) Press [RST] [2nd] [Op] **08** and the printer lists all labels and their absolute addresses. By using this feature you won't have to search through entire program listings to find where your labels are located.

# Index

# Index

(continued)

# Index

(continued)

# Index

(continued)

(continued)