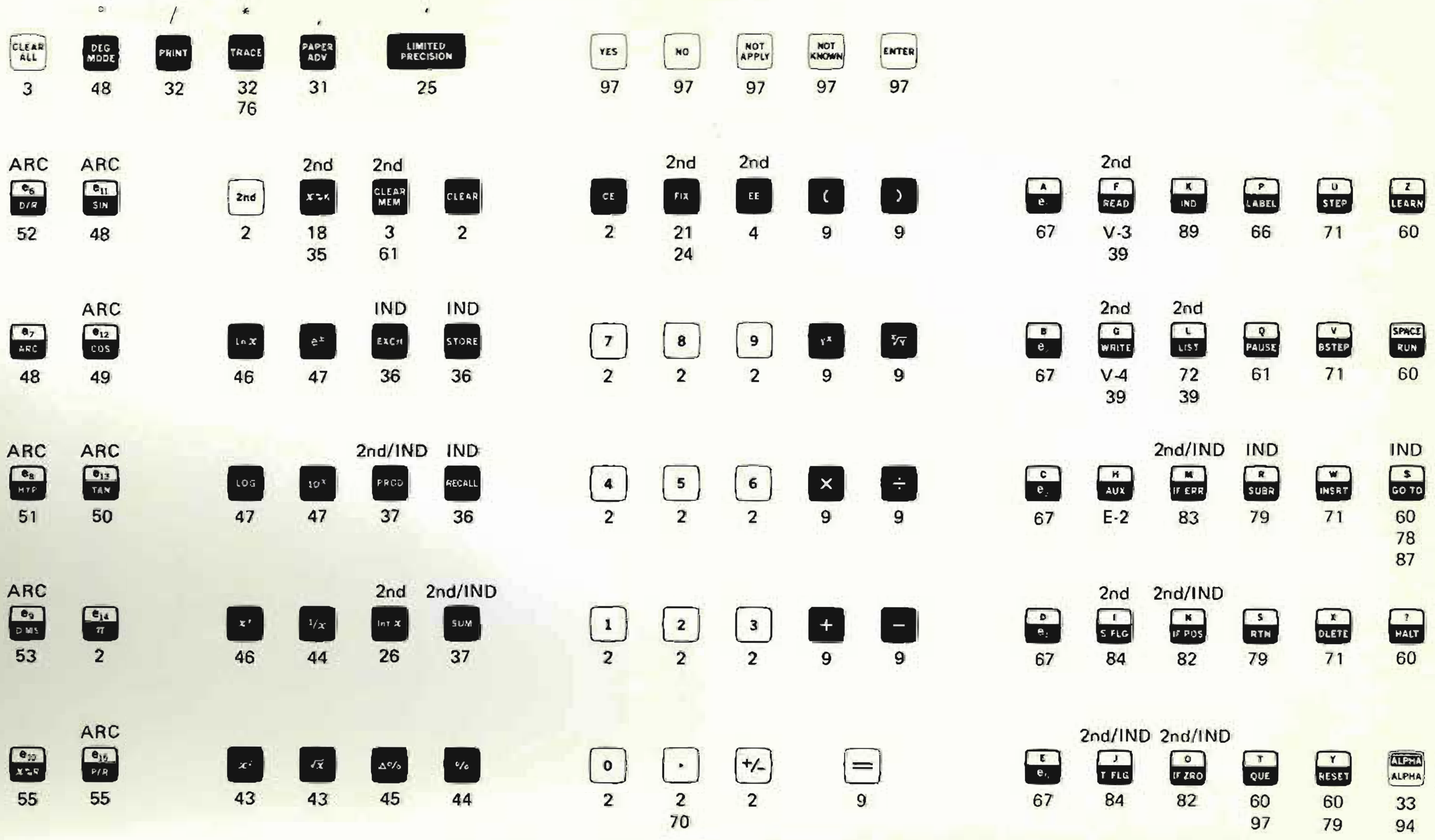
Texas Instruments
personal
computer/calculator
SR-60A

Operating/Programming Manual

KEY DESCRIPTION INDEX

This indexed keyboard provides a quick page reference to the description of each key in Section IV.



Texas Instruments

personal computer/calculator

SR-60A

Operating/Programming Manual

IMPORTANT

Record the serial number from the bottom of the calculator 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.

SR-60A		
Model No.	**Serial No.**	**Purchase Date**

This book was developed by:

Roger F. Farish
Staff of the Texas Instruments Learning Center
Dr. Ralph A. Oliva, Educational Software Director

With contributions by:

Michael Y. Brill
Robert E. Crocker
John F. Culhane
William L. Kennedy, Jr.
Henry M. Meltzer
Arthur L. Norrington, Jr.
Sydney W. Poland

Typesetting, Artwork and Layout coordinated and executed
by
Marca Advertising Inc.
Richardson, Texas

ISBN 0-89512-013-5

Library of Congress Catalog Number: 77-85370

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Section Page

I GETTING ACQUAINTED

INTRODUCTION

The SR-60A personal computer/calculator (referred to as calculator from here on) from Texas Instruments is designed to bridge the gap between simple desktop calculators and commercial computers. The advanced technologies of MOS integrated circuits, light-emitting diodes and electronic printheads are combined in the SR-60A to provide the functional benefits of a computer with the usage simplicity of a calculator.

Handles Like a Calculator

While the size of the SR-60A is comparable to a standard office typewriter, it weighs only 16 pounds. The keys are functionally grouped so that specific keys may be found without memorizing the location of all 95 keys. The *AOS*™ entry method and parentheses permit entry of a complex problem in the same left-to-right order in which the problem is normally written. Calculation results are displayed with as many as ten digits plus two digits for power-of-ten exponents. A quiet electronic printer can print operations and results on thermal paper for a permanent record of calculations.

More Functional Than a Computer

The programming functions of the SR-60A allows operations to be performed much like large computers except the intermediate steps – punching cards, compiling language decks, waiting for final results – are virtually eliminated. The programming language for the SR-60A is English. Not only are the key identifications in English, but the display and printer are capable of presenting the complete alphabet plus punctuation marks and other symbols. The SR-60A is a "prompting" calculator. Like a properly programmed computer, the calculator displays words or phrases to prompt or lead you in making entries or decisions when solving a specific problem. The printer identifies pertinent data and results with words so there is no confusion in analyzing the results.

Libraries of Prerecorded Programs

Many programs have already been written for your calculator and grouped into libraries of related programs. These programs have been placed on magnetic cards so that they are immediately available to you. You can begin using the full capabilities of your calculator without learning the mathematical or programming details associated with it.

CAUTION

Do not connect power cord until you have read Appendix A.

Getting Acquainted I

FEATURES AND FUNCTIONS

- **Full Prompting Capabilities** provide easy usage of many prewritten programs designed for your immediate use. You do not need computer experience. With only 5 minutes worth of knowledge about the SR-60A, this beneficial library of programs can be working for you. The calculator itself specifically asks you for each bit of information it needs to run the program.

- **Algebraic Operating System** (AOS) allows you to enter mathematical expressions in the same order that they are algebraically stated. Parentheses, an integral part of *AOS*, ensure proper and accurate interpretation of expressions. Up to 9 parenthesis levels with 10 pending operations are available.

- **Complete Set of Mathematical Functions** including:
  - Arithmetic Functions with algebraic hierarchy
  - Trigonometric Functions (including inverse functions)
    - Angles measured in degrees or radians
  - Hyperbolic Functions (including inverse functions)
  - Logarithmic Functions (both natural and common) with 10^x and e^x
  - Factorial, Reciprocal, Percent and Change of Percent
  - Square and Square Root, y^x and $\sqrt[x]{y}$
  - Pi (π) accurate to 12 digits
  - Constant feature for easy execution of repetitive calculations
  - Conversions for degrees to radians, degrees (hours)-minutes-seconds to decimal degrees (hours, polar to rectangular to spherical coordinate system and their inverses.

- **Complete Display Versatility**, featuring:
  - Standard 10-digit or 20 character display
  - Scientific Notation entry from keyboard and automatically from calculations
  - Scientific Notation removal
  - Fix Decimal control to select desired number of decimal places in the displayed number
  - Display value accuracy ensured by internal rounding
  - All results are calculated with 12 digits and rounded to obtain the displayed values
  - Limited Precision to select the accuracy of results

- **Automatic Clearing** — when the equals key is pressed, all calculations are completed, the answer is displayed and the calculator is ready for the start of a new problem.

- **Variable Storage Area for Programs** — Up to 2640 program locations in basic SR-60A (7920 in fully expanded version)
  - 15 User-Definable Keys
  - 12 Subroutine Levels
  - Transfer Instructions (3 different types)
  - Flags for path tracking and tagging
  - 152 available labels for identifying program parts
  - Complete edit capabilities
  - Magnetic card input and output

- **Variable Storage Area for Data** – Up to 330 registers in the basic SR-60A
  Store and Recall from any register
  Complete Memory Arithmetic directly into any register
  Memory/Display Exchange
  Magnetic Card input and output
  Data registers can also hold alphanumeric messages
- **Optional Memory Expansion** – the SR-60A can be easily modified to greatly increase storage capacity.
- **Alphanumeric Capabilities** – total versatility to:
  Label calculations
  Label program input and output
  Prompt you from a program so that you and the SR-60A actually converse, in English
  Plot data or results with any symbol.

  These capabilities can be easily *edited, stored in data memory* for use at any time and/or *listed from a program* just as they would be seen when displayed from a program.
- **Magnetic Card Permanent Storage** of any program or data set – 960 program steps or 120 data register values can be placed on each magnetic card for later use.
- **Printing Capabilities** for visual records
  Quiet thermal printer
  Prints displayed value any time you request it either from the keyboard or from a program
  Can trace a calculation sequence by printing the result of each step.
- **Auxiliary Equipment** – The SR-60A is designed to interface with other equipment
  CPT *Selectric* Typewriter
  Single or Dual Cassette Tape
  EIA (RS-232C) Interface
- **Library of Prerecorded Programs** ready for your immediate use. No programming experience and only a minimum of calculator familiarity necessary to use these programs.

HOW TO USE PRERECORDED PROGRAMS

You do not need prior experience in programming or even experience in using a calculator to be able to start using powerful programs. You simply choose one of our programs that suits your needs and supply the information needed to solve the problem itself.

What is a Prerecorded Program

First, consider the definition of a program. A *program* is a series of instructions or keystrokes that solve a specific problem. A prerecorded program, then, is a program that has been previously recorded on a magnetic card and is ready for your use. When the magnetic card is passed through the card-reading unit in the calculator, the instructions or keystrokes prerecorded on the card are read and remembered. The calculator automatically starts the program (performs or executes the keystrokes) to solve the problem, stopping only for you to make pertinent decisions or enter required data. It is important to note that the calculator only remembers a specific program until the power is turned off, the **CLEAR ALL** or **2nd, CLEAR MEM** key is pressed, or another prerecorded program is read from a magnetic card. Therefore, the magnetic card is the permanent record of a program and should be treated with care. Refer to *Caring for Magnetic Cards* in Appendix B, Maintenance and Service.

What Prompting Means

The SR-60A is a programmable calculator which has prompting capabilities. The use of the word "prompting" stems from the ability of the calculator to prompt you when running a program. It can display and print messages with up to 20 letters and numbers at a time asking you to respond with either the **YES, NO, NOT APPLY, NOT KNOWN,** or **ENTER** key. Thus, when a program is running and some unknown quantity is needed to complete the program, the calculator will stop and prompt you to enter the unknown quantity by displaying a statement such as "ENTER INTEREST(%)". At this point you simply key in the interest rate, press the **ENTER** key and the program continues. In some cases you are not allowed to use **NOT APPLY** or **NOT KNOWN**, depending on the program.

Another example of prompting is when a running program requires a decision to be made by you. In this case, you are prompted by the calculator with a question, such as "NEW VALUE OF X?". When this occurs, you have the option to use one of two answer keys, **YES** or **NO**.

The most frequent prompting message you will see is "PROMPTING DESIRED?". This question is displayed each time the power is turned on and when the **CLEAR ALL** key is pressed. If you answer with the **NO** key, the display goes to zero and the calculator is ready for manual calculations. If you answer with the **YES** key, indicating you wish to run a program, you are prompted with the instructions "PUSH YES, LOAD CARD". When the **YES** key is pressed the second time, the magnetic card-reading unit starts running so the prerecorded program can be read by the calculator when the magnetic card is loaded.

With the exception of the prompting messages described in the previous paragraph, the messages displayed and printed are actually developed from the prerecorded programs. The originator of the program is responsible for programming the messages and making them fit within the 20-character per message allotment. This point is made to help you understand that the abbreviated way some messages may appear is a result of the number of characters that can be displayed at one time.

Running a Program

To run a prerecorded program, you need two items in addition to the calculator. You must have the prerecorded magnetic card containing the program you wish to run, and the user instructions for the program found in the manual for that library. User instructions are just what the name implies, a set of instructions to tell you (the user) in as few words as possible, how to use the program. The following figure illustrates a completed user instructions form developed for the SR-60A. These user instructions are for the Compound

Interest program from the Basic Library provided with the calculator. This program will be used in the following description; therefore, please locate the magnetic card furnished with the calculator entitled COMPOUND INTEREST.

Referring to the User Instructions below, notice there are five columns: STEP, DISPLAY, INSTRUCTION, PRESS and GO TO. The STEP column simply provides a sequence of the major events of the program. The DISPLAY column shows the prompting messages to be displayed by the calculator during the program. The INSTRUCTION column provides itemized instructions on the decision or response required for the message in the DISPLAY column. The PRESS column shows the key to be pressed that corresponds to the selected instruction. If the user instructions are not performed sequentially from one step to the next, the GO TO column shows the step number you should proceed to if the key in the adjacent PRESS column is pressed.

STEP	DISPLAY	INSTRUCTION	PRESS	GO TO
1		Load card 2 side B		
2	ENTER PRESENT VALUE	a. If present value is known, key it in b. If unknown	ENTER NOT KNOWN	
3	ENTER FUTURE VALUE	a. If future value is known, key it in b. If unknown	ENTER NOT KNOWN	
4	ENTER INTEREST (%)	a. If interest rate per period is known, key it in b. If unknown	ENTER NOT KNOWN	
5	ENTER NO. PERIODS	a. If number of periods is known, key it in b. If unknown	ENTER NOT KNOWN	
6		Value of the unknown variable is computed and printed.		2

Note: Interest is entered and printed as a percentage. For example, 9% is entered as 9.

User Instructions Form

Now consider running the COMPOUND INTEREST program. For explanation purposes, let's find the future value of $500 in a savings account after five years if the interest rate is 6%, compounded annually.

The first step of the user instructions is to Load Card 2 Side B. This instruction to load a card is the initial instruction for each program and the loading procedure is described below and in the front of the Program Manual of each library.

Getting Acquainted

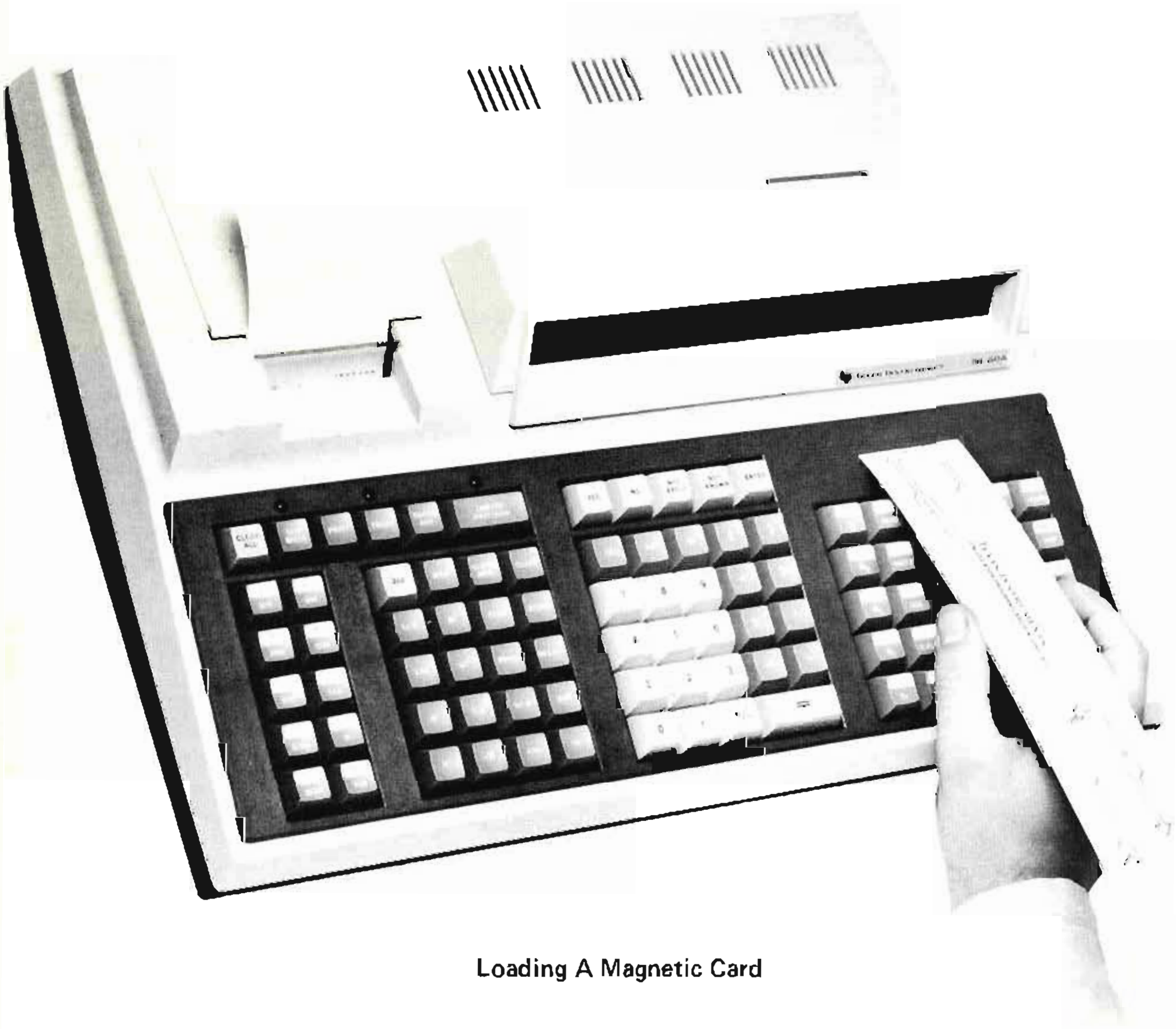
I

1. Turn calculator ON or press the **CLEAR ALL** key. Display shows "PROMPTING DESIRED?".

2. To run a program, press the **YES** key. Display shows "PRESS YES, LOAD CARD".

3. Press the **YES** key again. Display goes blank and card-reader motor begins running.

4. Insert the end of the magnetic card into the card-reader slot with the arrow corresponding to the desired program title pointing toward the slot as shown below. Gently feed the card into the slot until it is pulled through the calculator by the motor. Do not hold or restrict card travel while it is engaged by the motor.



Loading A Magnetic Card

5. If you are running a program which uses only one side of a card, proceed to step 6. Otherwise, continue with this step. After the motor stops, the display will show "PRESS YES, LOAD CARD". Locate the next card in the sequence, press **YES** and repeat step 4. Repeat procedure until all cards are entered. DO NOT USE THE **CLEAR ALL** KEY UNLESS IT IS NECESSARY TO RESTART THE LOADING PROCEDURE.

6. After the motor stops, remove card and return it to its holder. The printer* should print the title of the program. Proceed to the second step of the user instructions. If after the motor stops, a flashing question mark appears on the display, restart this loading procedure with step 1. If difficulty persists, refer to *In Case of Difficulty* in Appendix B, Maintenance and Service.

   *Some programs do not automatically engage the printer. Proceed to the second column of the user instructions for normal display indication.

Following this procedure should complete step 1 of the user instructions. The display shows "ENTER PRESENT VALUE", indicating the program has been running and has stopped at step 2. To avoid confusion, the user instructions do not show the printout of a program. The figure below represents the printout of the present problem with the result of each step indicated by number.

By observing the response choices in the INSTRUCTION column, the correct action is to key in the present value (500) and press the ENTER key. The calculator now displays "ENTER FUTURE VALUE" which indicates the program is at step 3. Since the future value is an unknown quantity, the only response is to press the **NOT KNOWN** key. Notice that the printer does not respond for this operation. The display now shows "ENTER INTEREST (%)" as indicated by step 4. Since this is a known value, key in the interest rate (6) and press the **ENTER** key. The display now shows "ENTER NO. PERIODS", indicating step 5. The last response is to key in the number of years (5), press the **ENTER** key and step 6 is automatically completed with the printout of the future value. Notice that the only entry in the GO TO column is following step 6, which indicates the program automatically returns to step 2 for you to begin another problem.

```
  **COMPOUND INTEREST

PRESENT VALUE =
          500.00
INTEREST (%) =
              6.
NO. PERIODS =
              5.

FUTURE VALUE
          669.11

***
```

Typical Program Printout

There are a few variations in the basic display format not illustrated by the previous problem which are used in other programs. Another method to ask for present value to be entered is "PRESENT VALUE =". The equals sign implies that a number should be keyed in and followed by the **ENTER** key. Another form of message commonly displayed is a word or words followed by a question mark. The response to a question should be either YES or NO.

Tips for Running Programs

As in using any machine or calculator, it is reasonable to expect that some errors might be made during initial operation. By reading the instructions carefully, you should soon be able to run programs with confidence. The following hints will help you eliminate most of the common difficulties.

First solve a known problem. It is good practice to run a program the first time using a problem that has a known answer. At least one example problem is included with each library program for this purpose.

An erroneous entry may be corrected before the **ENTER** key is pressed by pressing the **CE** key. Entry errors discovered after the **ENTER** key has been pressed cannot normally be corrected. Press the **QUE** key to restart the program. You do not need to reload the program card(s). Some programs are specially written to ask for deletions or corrections. The user instructions should be consulted in these cases.

Intermediate calculations. Most library programs allow you to perform manual calculations after they stop with a prompting message. This allows you to precalculate data for the program when you have it in a different form. For instance, you can convert an annual interest rate to a monthly rate. The appropriate response key must then be pressed to correctly continue the program. A simple test can be made to determine if intermediate calculations will affect the program. When the prompting message appears and intermediate calculations are desired before continuing the program, press the **CE** key and observe the displayed number. Next press the equals key. If the displayed number does not change, intermediate calculations are possible. Otherwise, press **QUE** to restart program.

The following keys should not be used in an intermediate calculation: The 30 programming keys on the right side, **CLEAR MEM**, **CLEAR ALL**, and **TRACE**, **DEG MODE** or **LIMITED PRECISION** unless status lights are observed and restored to same status before continuing program, and the **STORE**, **EXCH**, **PROD** and **SUM** keys without first checking for which registers are used by the program.

Wrong answers or flashing display. Typical reasons are: Entries which force the calculator into overflow, underflow, or an error condition – check for limits in program description. Pressing the **ENTER** key without making a number entry. Incorrect number entry. Program error – refer to *In Case of Difficulty* in Appendix B, Maintenance and Service.

II "A GUIDED KEY TOUR"

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 into 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 III).

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

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

When the calculator is first turned on, the display asks, "PROMPTING DESIRED?" – meaning, do you want to run a prerecorded program? Pressing any key on the keyboard tells the calculator NO. Now manual calculations can begin.

KEYBOARD BASICS

CLEARING OPERATIONS – CE, CLEAR, CLEAR ALL

There are several procedures for clearing your calculator depending upon your needs as you proceed through a problem.

CE CLEAR ENTRY – This 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 and remove the question mark created by an error condition.

CLEAR GENERAL CLEAR – This 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.

CLEAR ALL CLEAR ALL – Completely clears the calculator including all data registers and program memory. Results in display message "PROMPTING DESIRED?" This is a master clear key that resets and clears everything. Be sure that this is what you want before you press this key.

A "Guided Key Tour" II

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.

Pressing π places the first 10 digits of π in the display, 3.141592654. Twelve digits are carried in the internal display register, 3.14159265359. CE does not remove this entry.

BASIC OPERATION KEYS – + , – , × , ÷ , =

Basic arithmetic is handled with the 5 basic operations: +, –, X, ÷, =. Your calculator has a powerful feature called the *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 equals 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 below.)

When you press the equals 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 + 8.231 – 0.08 = ?

Press: 15 + 8.231 – .08 =   Display: 23.151

Certain simple arithmetic calculations can be stored as a constant with the X ⪌ K key and used repeatedly, saving many keystrokes. See *Calculations with a Constant Number* in Section IV.

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 X 14 ÷ 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 hierarchy of the calculator sorts the operations you enter, applies them in the correct order, and lets you see what it's doing along the way. Your calculator performs operations it received from you in the following universally accepted order:

II

A "Guided Key Tour"

1) Algebraic Functions – 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, percent and conversions.)

2) Percent Difference (Δ%)

3) Powers and Roots (y^x and $\sqrt[x]{y}$) are handled next

4) Multiplications and Divisions are completed, followed by

5) 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.

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 10 operations pending. The following is an example of this full capacity.

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

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
CLEAR	0	Clear any calculations in progress
(8 × (4 + 9)	13.	(4 + 9) is evaluated
+	104.	8 X (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

MEMORY KEYS — CLEAR MEM , STORE , RECALL , EXCH

Each time you turn on your calculator there are 100 data registers available (expanded machines have more) 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 IV for complete details.) Data registers are special locations in the calculator where you can store numbers or alphanumeric characters you may need to use later.

Because there is always more than one data register available for your use, you must indicate which register you want to use by specifying its address number N. For example, **STORE** 80 stores the displayed value in register 80.

The **CE** and **CLEAR** keys do not affect the contents of the memories; however, pressing **CLEAR MEM** clears all data registers simultaneously (places a 0 in all registers).

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

RECALL N – **RECALL** – This instruction simply brings the contents of data register N to the display register. The contents of data register N are not disturbed.

EXCH N – **MEMORY EXCHANGE** – The exchange sequence simply swaps whatever is in data register N with the contents of the display register. (The display register value is stored in register N 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 STORE 09	3.21	Store 3.21 in register 09
CLEAR	0	Clear display
RECALL 09	3.21	Recall contents of register **09**

You do not need to enter the leading zeros of a memory address if the address is immediately followed by a nonnumeric key. If no number is entered for an address, it is assumed to be register 00.

Example: Store 5 in register 12 and 4 in register 00, then add the two together.

Press	Display	Comments
5 STORE 12	5.	Store 5 in register 12
4 STORE +	4.	Store 4 in register 00
RECALL 12 =	9.	Add 4 to contents of register 12

If you have an expanded calculator, the first step should be **5, STORE 012**, because there are more than 100 data registers available.

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

Press	Display	Comments
CLEAR	0	Clear any calculations in progress
9.3069128 STORE 12	9.3069128	Stores A in register 12
+ 2 +	11.3069128	A + 2 is evaluated
EXCH 12	9.3069128	Stores A + 2 in register 12 and calls A to the display register
× RECALL 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 , PROD

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 N – **MEMORY SUM** – This sequence allows you to add whatever is in the display register directly to the number stored in register N. The result of the addition is stored in the memory while the display register is unaffected. Similarly, the sequence **2nd, SUM, N** subtracts the value in the display register from the contents of register N.

PROD N – **MEMORY PRODUCT** – This sequence causes the contents of register N to be multiplied by the display register value while **2nd, PROD, N** divides the value in register N 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 basic arithmetic operations in normal keyboard calculations, except that results are accumulated in a data register instead of the display register. The results stored in the data registers are carried to 13 digits.

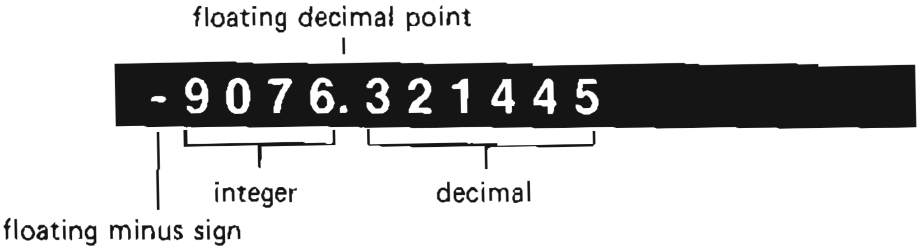
Example: Find the total cost of items of $28 and $6.60 with 5% sales tax.

Press	Display	Comments
28 STORE 01	28.	Store 28 in data register 01
6.6 SUM 01	6.6	Add 6.6 to data register 01
1.05 PROD 1	1.05	Multiply data register 01 by 1.05
RECALL 01	36.33	Total Cost

DISPLAY CONTROL

STANDARD DISPLAY

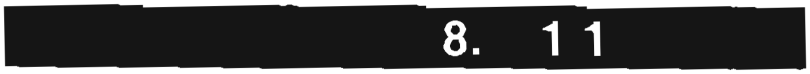
The display provides numerical information complete with negative sign and decimal point and flashes on and off for an overflow, underflow, or error condition. (A complete list of error conditions is found in Appendix C.) An entry can contain as many as 10 digits. All digits entered after the tenth are 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 12 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. X 10^{11}.



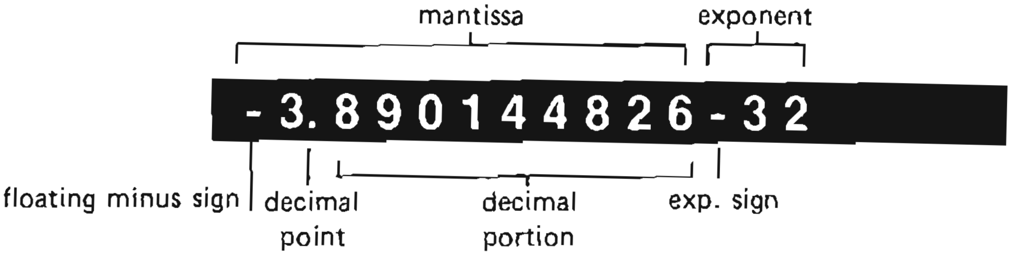
SCIENTIFIC NOTATION KEY – EE

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 10 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 $-3.890144826 \times 10^{-32}$ looks like this in your display:



Note: DO NOT enter "X 10." Correct: –3.8901 EE –32, not –3.8901 X 10 EE –32.

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 scientific notation format, the display stays in that format until you deliberately remove it. When you press **2nd**, **EE**, the calculator returns to standard display format as soon as the value in the display is within the range of the standard display. **CLEAR** removes this format when it clears the display.

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 **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 entries with as many digits as you like as the calculator retains its own internal (12-digit) accuracy. **FIX 9** removes fix-decimal format.

Example: 2 ÷ 3 = .6666666667

Press	Display
CLEAR	0.
2 ÷ 3 =	.6666666667
FIX 6	0.666667
FIX 2	0.67
FIX 0	1.
FIX 9	.6666666667

INTEGER AND DECIMAL FRACTION CONTROL – Int x

It is occasionally desirable to conveniently discard all digits to the right or left of the decimal. The **Int x** key will quickly perform this operation.

Int x INTEGER KEY – Instructs the calculator to discard the decimal-fraction digits of the number in the display and in the display register. The integer operation is performed directly on the display register and retains the true integer value without regard to display format.

2nd Int x INTEGER REMOVAL – Instructs the calculator to discard only the integer digits with the conditions specified for the integer key.

Example: Determine the integer part of 31 ÷ 9.

Press	Display
CLEAR	0
31 ÷	31.
9 =	3.444444444
Int x	3.

Example: Determine the decimal-fraction portion of 3.59463 X 10^3.

Press	Display
3.59463 EE	3.59463 00
3	3.59463 03
2nd Int x	6.3–01

ROUNDING AND LIMITED PRECISION

Normally, the calculator *rounds the value in the display register* (that can contain up to 12 digits) so that it can be displayed in the 10 available positions in the display window. The whole display register value, though, is used for all further calculations — the rounding is for display purposes only. The value is rounded up (one added to the rightmost digit), if the next, nondisplayed digit is 5 or more. Otherwise, the extra digits are just held in the display register, awaiting further calculations.

This normal procedure can be altered according to various needs through use of **2nd**, FIX and **LIMITED PRECISION**. For details of these advantages, see *Rounding Control and Limited Precision Control* in Section IV.

ALGEBRAIC FUNCTIONS

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

SQUARE, SQUARE ROOT, RECIPROCAL, FACTORIAL KEYS – x^2, $\sqrt{x}$, 1/x, x!

x^2 – SQUARE – Calculates the square of the number, x, in the display register.

$\sqrt{x}$ – 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.

x! – FACTORIAL – Calculates the factorial of the number, x, in the display register.

Example: $\sqrt{4} \div (1/5)^2 = 50$

Press	Display	Comments
CLEAR	0	Clear any previous calculations
4 $\sqrt{x}$	2.	$\sqrt{4}$
÷ 5 $1/x$	0.2	1/5
x^2	0.04	$(1/5)^2$
=	50.	The result

POWERS AND ROOTS – y^x , $\sqrt[x]{y}$

These powerful keys allow you to raise a positive number to a power or find the root 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
CLEAR	0
2 y^x 6 =	64.

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

- Enter the number (y) you want to find a root of
- Press $\sqrt[x]{y}$
- Enter the root (x)
- Press = (or any operation key)

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

Press	Display
CLEAR	0
64 $\sqrt[x]{y}$ 6 =	2.

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

PERCENTAGES – % , Δ%

% PERCENT – Instructs the calculator to convert the number in the display register from a percentage to a decimal value (moves decimal point two places to the left). When used after an addition or subtraction operation, add-on or discount calculations may be made. When used after other operations, percentages are converted to decimal values for further calculations.

Example: Calculate the price of a $15.95 item that is discounted 20% and include a sales tax of 5%.

Press	Display
CLEAR FIX 2	0
15.95 −	15.95
20 %	3.19
+	12.76
5 %	0.64
=	13.40

Δ% PERCENT CHANGE – Instructs the calculator to determine the percentage change between two values. The first value is the data entry or calculated result being displayed when Δ% is pressed. The second value must be a data entry. Pressing the equals or another operation key completes the calculation and causes the percent change to be displayed.

Note: The percent change is calculated in the following manner:

$$\frac{(\text{second value} - \text{first value}) \times 100}{\text{first value}} = \text{percent change}$$

Example: A $5 item is marked down to $3. What is the discount percentage?

Press	Display
FIX 9 CLEAR	0
5 Δ%	5.
3 =	−40.

LOGARITHMS – Ln x , LOG , 10ˣ , eˣ

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 = 2.71828182846) of the number held in the display register. (A flashing display results if this number is negative or zero.)

LOG COMMON LOGARITHM — Immediately calculates the common logarithm (base 10) of the display register value (again, the value in the display must be positive).

10^x COMMON ANTILOGARITHM — Raises 10 to the x power, essentially findingthe common antilogarithm of x.

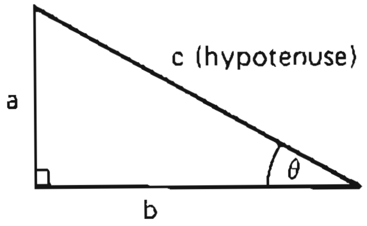
e^x NATURAL ANTILOGARITHM — Raises exponential e to the x power, essentially finding the natural logarithm of x.

ANGULAR MODES

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 or radians. Your calculator always powers up in the radian mode; however, you may select degrees for angular measure by pressing the **DEG MODE** key. A red light above the key indicates that you are in the degree mode.

TRIGONOMETRIC KEYS — **e_{11} SIN**, **e_{12} COS**, **e_{13} 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.



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

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

The sequences **ARC SIN**, **ARC COS**, and **ARC TAN** are used to calculate the inverses of these functions. For example, arc sine of θ is "the angle, θ, whose sine is x". Arc sine is also written "$\sin^{-1}$" or "inverse sine". The resulting angles are displayed in units corresponding to the selected angular mode.

In the degree mode, all angles are interpreted in decimal format. (See *Degree Format Conversions* in Section IV.)

HYPERBOLIC FUNCTIONS – HYP SIN , HYP COS , HYP TAN

These sequences calculate the hyperbolic sine, cosine and tangent of the value in the display register. Pressing the **ARC** key before any of these sequences will result in the inverse hyperbolic sine, cosine or tangent being taken of the display register value.

CONVERSIONS

The SR-60A is capable of performing several intricate conversions simply by touching the correct key. Probably the most useful is the conversion of degrees-minutes-seconds or hours-minutes-seconds to decimal degrees or hours. There are also keys dedicated for conversion between degrees and radians and between rectangular, polar and spherical coordinate systems. More about these in Section IV.

BASIC PRINTER OPERATIONS – PAPER ADV , PRINT , TRACE

The printer is built into the SR-60A to provide you with a permanent record of your calculations. When running prerecorded programs, the printout tape is the primary source of data and results while the display normally presents prompting messages and instructions. The printing itself is done by a thermal print head that quietly activates heat sensitive paper. Other heat sources, such as radiators, prolonged sunlight or hot cigarette ashes may also activate the heat sensitive paper.

PAPER ADV PAPER ADVANCE – This key advances the printer paper without printing. If the key is pressed quickly, a single unwritten line is advanced. If the key is held down, the paper will continue to advance until the key is released. The paper advance instruction can be placed in a program as well as used from the keyboard.

PRINT PRINT – This key causes the current contents of the display to be printed. If the content of the display is an alphanumeric message, it will be cleared from the display after printing. This instruction can also be used in a program.

TRACE TRACE – This key causes the calculator to enter the trace mode. In this mode every new function or result is automatically printed. Number entry keys do not cause a line to be printed. A number entry followed by a function will cause a line to be printed. When the **TRACE** key is pressed, an indicator light comes on above the key and remains on until the key is pressed again, that takes the calculator out of the trace mode.

When in the trace mode of operation, the printer provides a detailed record of numbers, function entries and results. Since the calculator must devote some amount of time to the printing process, it will ignore keyboard entries during the short printing periods following function entries. **Be careful not to make entries while the printer is operating**, because they will be ignored.

Example: Use the trace mode to print out the following calculation.

2.65 + 3.95 = 6.6

Press	Display	Printout
CLEAR ALL NO	0	
TRACE	0	0
2.65 +	2.65	2.65 +
3.95 =		3.95 =
	6.6	6.6
CLEAR	0	6.6 CLR
TRACE	0 TRC	0 TRC

The printer has more capabilities. See *Printer Operations* in Section IV.

III 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 calculator 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. The language that allows you to communicate with your SR-60A is English, making programming as simple as possible.

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 III

ELEMENTARY PROGRAMMING

PLACING A VARIABLE IN A PROGRAM

Consider the following simple expression:

$$A + B = C$$

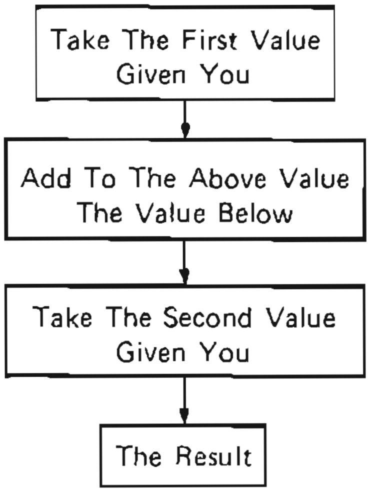
When using a basic four-function calculator, the values of A and B cannot be identified at a later time, they must be known at the time the expression is keyed in. After this first expression is keyed in and a result is obtained, to change either or both values you have to key in the entire expression 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.

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

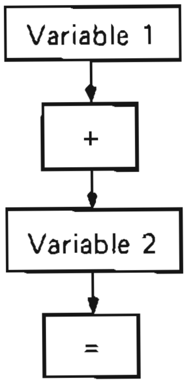
You'd like to be able to enter the equation just once, then change only the value of A for each calculation. With your programmable calculator it's easy to do just that.

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.



Take The First Value Given You
Add To The Above Value The Value Below
Take The Second Value Given You
The Result

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

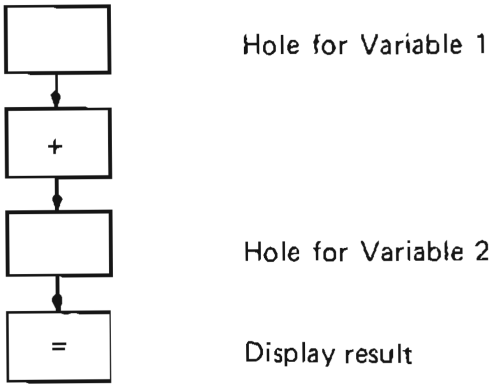


As far as the calculator is concerned, if you don't enter the variable as part of the program, two things must be done:

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

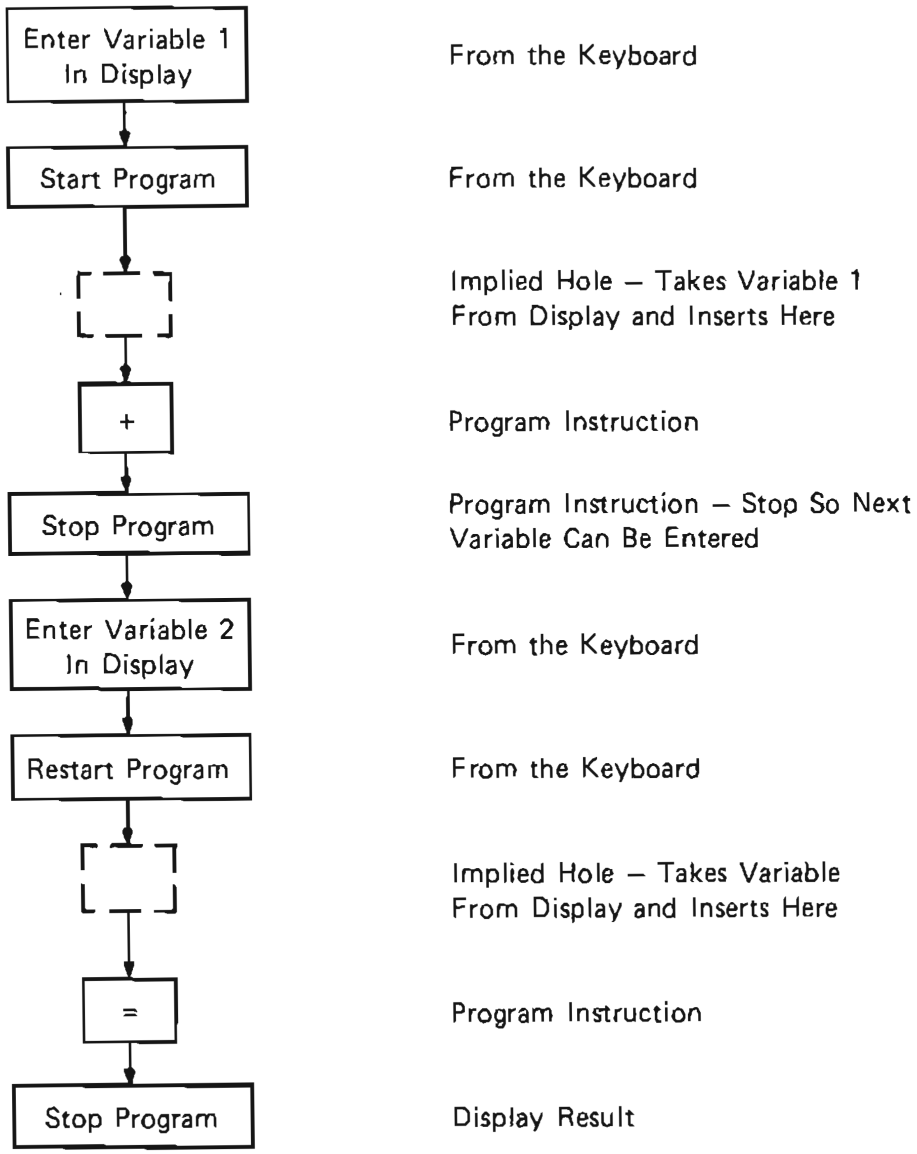
2. Tell the calculator where to look for the variable value when it needs it. You can instruct the calculator to look for a variable either in the display or in one of its data registers.

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.

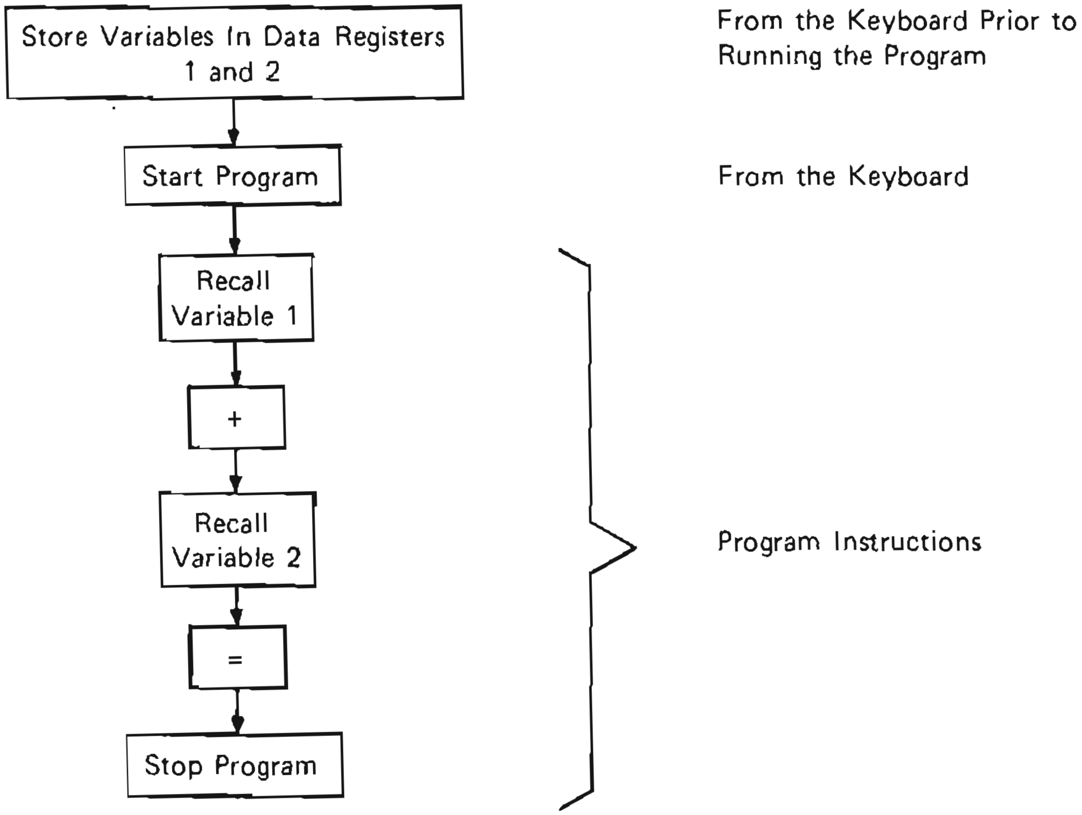
One technique that lets you leave holes in your program for new entries or data is to simply stop the program at that point with the HALT key. You're just telling the machine to hold everything so that you can key the next value it needs into the display. (We can call this an implied hole since no gap 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.)


Enter Variable 1 In Display
From the Keyboard
Start Program
From the Keyboard
Implied Hole – Takes Variable 1 From Display and Inserts Here
+
Program Instruction
Stop Program
Program Instruction – Stop So Next Variable Can Be Entered
Enter Variable 2 In Display
From the Keyboard
Restart Program
From the Keyboard
Implied Hole – Takes Variable From Display and Inserts Here
=
Program Instruction
Stop Program
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.

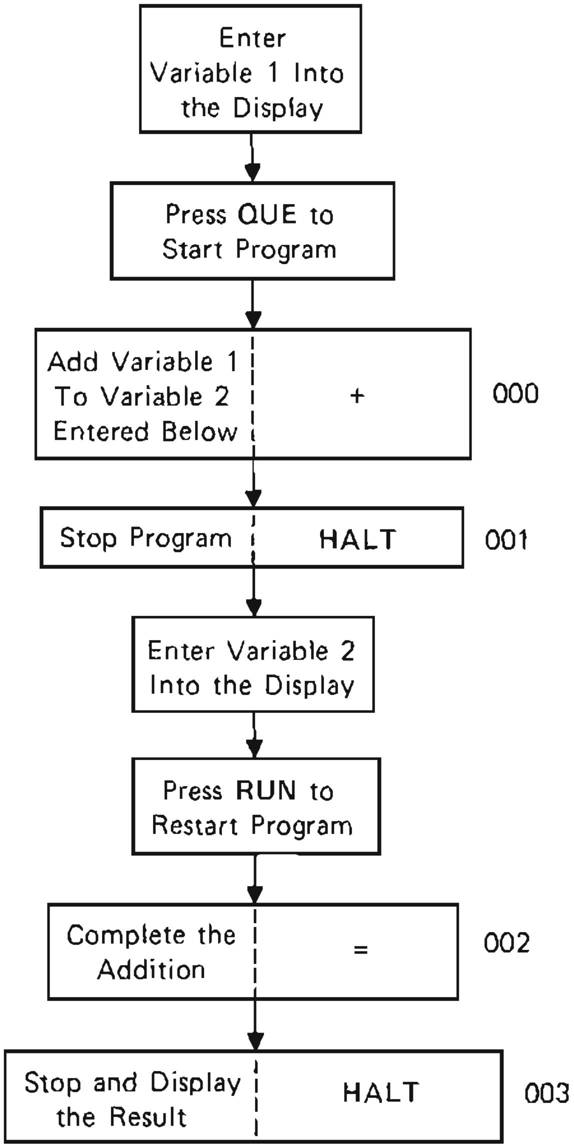
If you want your calculator to find a variable in its memory, place the instruction to recall the variable from the appropriate data register right in your program. For example, recalling a variable stored in data register 1 is performed by the sequence **RECALL 01**.



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. The **QUE** key is usually used to start a program. When pressed from the keyboard, it causes the program to begin running at the start (program memory location 0000). The following example shows the keystrokes required to instruct the calculator to look for the variables in its display register.


Enter Variable 1 Into the Display
Press QUE to Start Program
Add Variable 1 To Variable 2 Entered Below
+
000
Stop Program
HALT
001
Enter Variable 2 Into the Display
Press RUN to Restart Program
Complete the Addition
=
002
Stop and Display the Result
HALT
003

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.

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

1. Press the **CLEAR ALL** key to completely clear the calculator, then press **NO** to tell the calculator that you don't want to read a card.

2. Press the **LEARN** key to enter the learn mode. You will know you are in this mode by a unique display format 0000 Δ.

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 **LEARN** key a second time to exit the learn mode and the unique display blinks, then disappears leaving a single zero displayed. You are now ready to run the program.

The four digits in the left of the display should change as you enter a program. These four 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 and press **NO**.

2. Press the **LEARN** key to enter the learn mode.

3. Enter the program by pressing the sequence + **HALT** = **HALT**.

4. Press the **LEARN** 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 **CLEAR** to clear any calculations in progress.

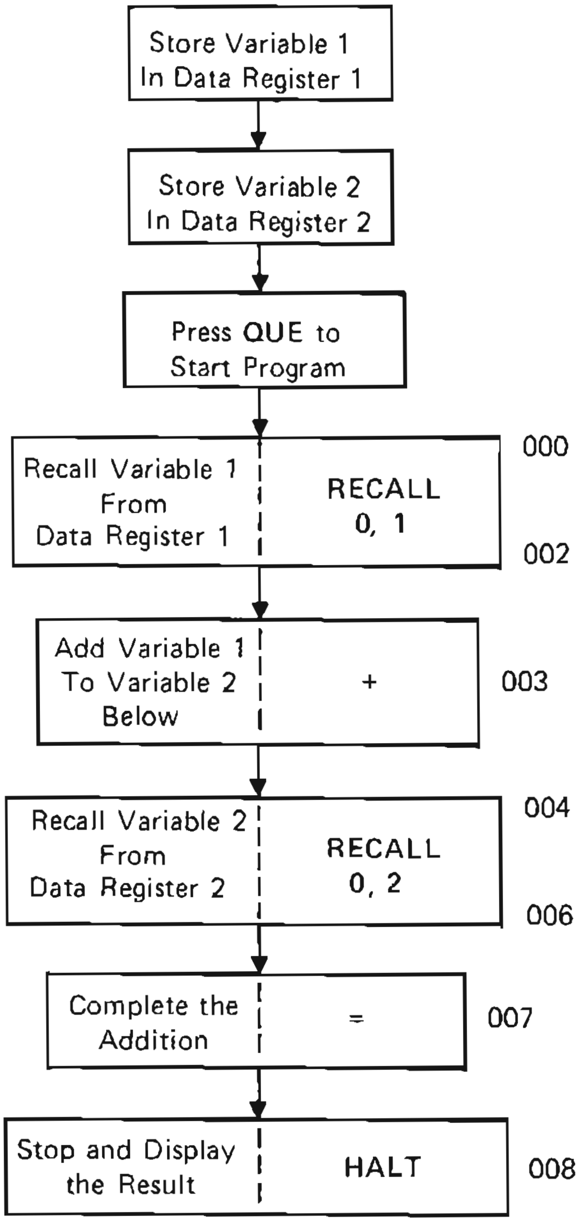
2. Enter **227** for variable 1.

3. Press **QUE**. The number **227** remains in the display.

4. Enter **34** for variable 2 and press **RUN**. The answer **261** is displayed.

Before running any program, it is good practice to press the **CLEAR** key to ensure that there are no pending calculations left to cause erroneous results.

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


Store Variable 1 In Data Register 1
Store Variable 2 In Data Register 2
Press QUE to Start Program
Recall Variable 1 From Data Register 1
RECALL 0, 1
000
002
Add Variable 1 To Variable 2 Below
+
003
Recall Variable 2 From Data Register 2
RECALL 0, 2
004
006
Complete the Addition
=
007
Stop and Display the Result
HALT
008

Variables in Data Memory Program

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

1. Press **CLEAR ALL, NO.**

2. Press the **LEARN** key to enter the learn mode.

3. Enter the program by pressing the following sequence.

```
RECALL
0
1
+
RECALL
0
2
=
HALT
```

4. Press the **LEARN** key again to exit from the learn mode.

This program looks for the variables in data registers 1 and 2. Therefore, store 227 in register 1 and 34 in register 2 as follows:

1. To Store 227 in memory 1
   Enter **227**
   Press **STORE 01**

2. To Store 34 in memory 2
   Enter **34**
   Press **STORE 02**

As noted in the flow diagram comments, the only keyboard operation needed to run the program is to press QUE. After pressing these keys, the answer **261** appears in the display and the problem is solved.

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.

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 III

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, CLEAR MEM, LEARN** and key in the complete program from the coding form. When entry is complete, press **LEARN** 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 **LEARN** to return to the keyboard operation.

10. **Retest the program.** Repeat steps 7-9 as needed.

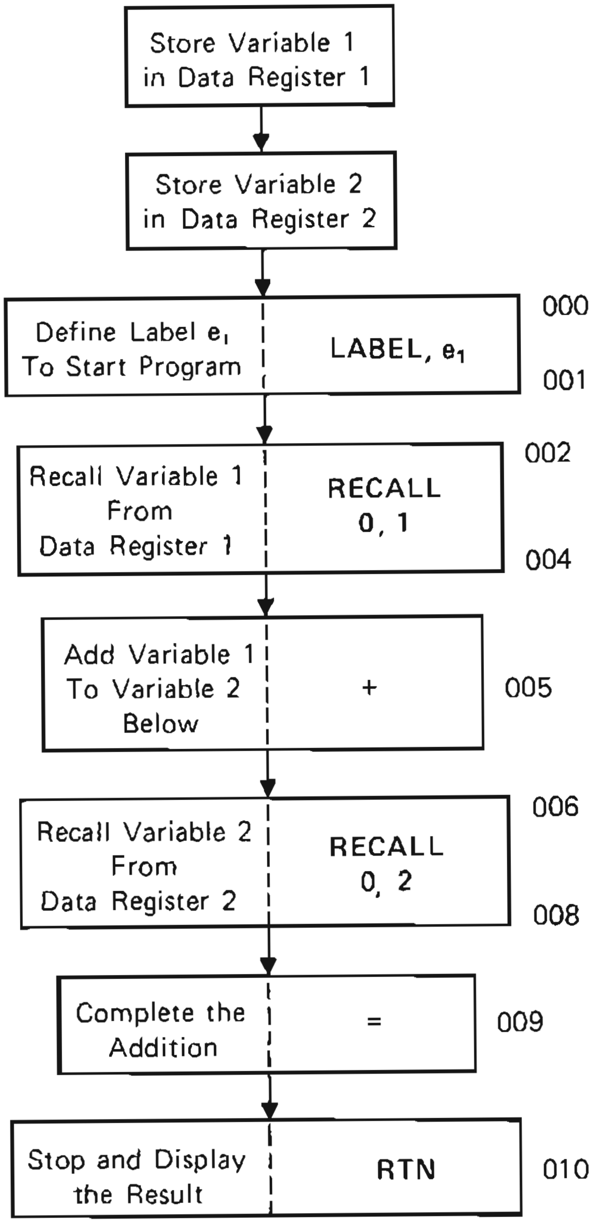
11. **Record the program.** Record the program on magnetic cards or list it out by pressing **RESET**, **LIST** if you wish to save it.

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.

USING USER-DEFINED KEYS (LABELS)

In running the previous sample programs, you used the **QUE** and **RUN** keys. Since **QUE** returns the program pointer to instruction number 0000, 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. These keys are **e_1** through **e_5** and **2nd, e_6** through **2nd, e_{15}** which allow you to identify and access up to 15 different reference points (programs or parts of programs).

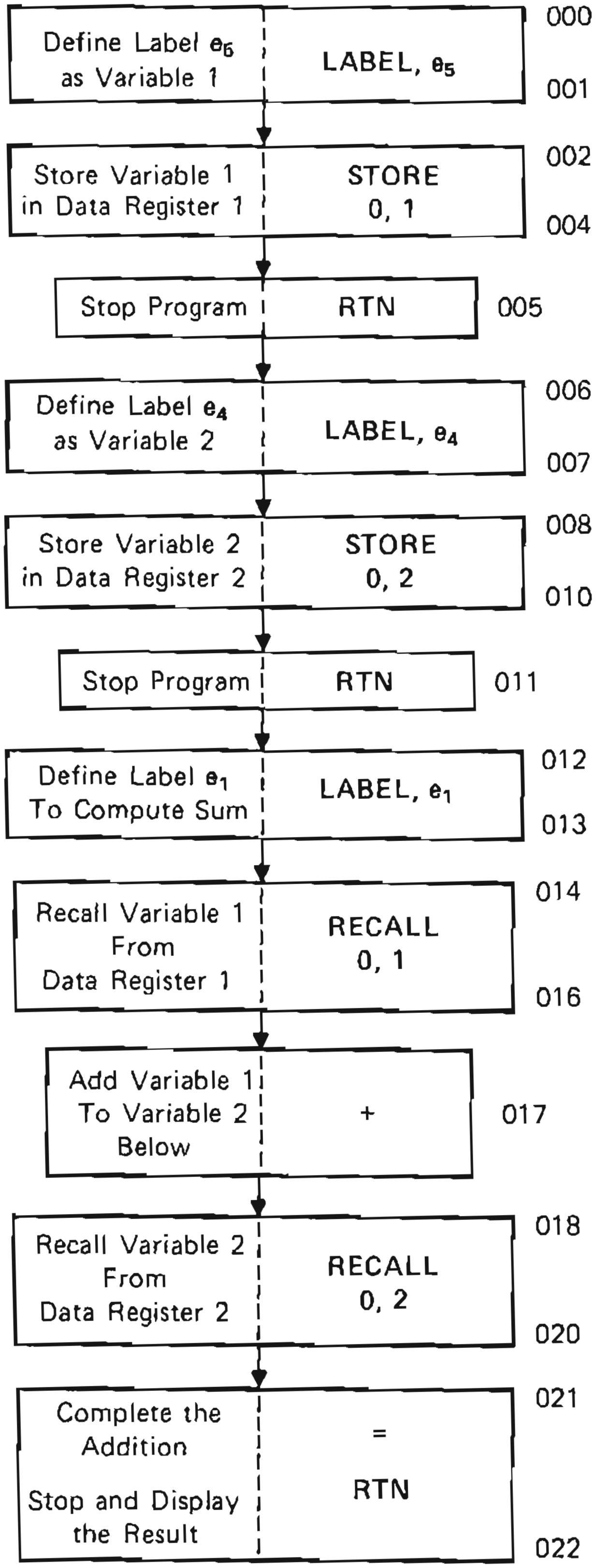
When a user-defined key is placed in a program, pressing this key causes the program pointer to locate the label. The calculator then automatically begins running the program, starting calculations from the first instruction following the label. For example, with a minor addition to the first program example, the key used to start the program could be **e_1** or any other user-defined key. As a matter of fact, if the program starts anywhere other than at location 0000, the **QUE** key could not be used to start the program. Since the **e_1** key is user-defined, the addition to the program is simply to label the start of the program with **e_1** using the **LABEL** key as shown in the following diagram.


Store Variable 1 in Data Register 1
Store Variable 2 in Data Register 2
Define Label e₁ To Start Program
LABEL, e₁
000
001
Recall Variable 1 From Data Register 1
RECALL 0, 1
002
004
Add Variable 1 To Variable 2 Below
+
005
Recall Variable 2 From Data Register 2
RECALL 0, 2
006
008
Complete the Addition
=
009
Stop and Display the Result
RTN
010

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 e_1. The answer "261" is displayed, because pressing e_1 tells the calculator to find where **LABEL** e_1 is located in program memory and then begin executing the instructions or keystrokes following **LABEL** e_1.

Notice that **RTN** is used to end the program instead of **HALT**. This key performs the same function as **HALT**, but with several advantages that are discussed later.

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 e_5 and variable 2 to be stored in data register 2 by pressing e_4. Then, e_1 is again used to obtain the answer.


Define Label e_5 as Variable 1
LABEL, e_5
000
001
Store Variable 1 in Data Register 1
STORE 0, 1
002
004
Stop Program
RTN
005
Define Label e_4 as Variable 2
LABEL, e_4
006
007
Store Variable 2 in Data Register 2
STORE 0, 2
008
010
Stop Program
RTN
011
Define Label e_1 To Compute Sum
LABEL, e_1
012
013
Recall Variable 1 From Data Register 1
RECALL 0, 1
014
016
Add Variable 1 To Variable 2 Below
+
017
Recall Variable 2 From Data Register 2
RECALL 0, 2
018
020
Complete the Addition
=
Stop and Display the Result
RTN
021
022

Place your calculator in the learn mode and enter the program instructions. Press **LEARN** 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 because it requires 4 less keystrokes than the other two versions.

First Version	Second Version	Third Version
Enter **227**	Enter **227**	Enter **227**
Press **STORE 01**	Press **STORE 01**	Press e_5
Enter **34**	Enter **34**	Enter **34**
Press **STORE 02**	Press **STORE 02**	Press e_4
Press **QUE**	Press e_1	Press e_1
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 **RECALL 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.

III

Programming Considerations

SHORT-FORM ADDRESSING

Until now we have always used a two-digit address (three-digit address for more than 100 data registers) to access data registers. That is, recalling a variable stored in data register 1, for example, has been accomplished by using the sequence **RECALL 01**. In some cases, however, leading zeros are not needed to access data registers. This type of addressing is called *short-form addressing* and may be used whenever a nonnumeric keystroke immediately follows the register address.

Example: Store 227 in data register 1 and 34 in data register 0, then recall these values and compute their sum.

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

Observe that when short-form addressing is used, the instruction is not complete until a nonnumeric key is pressed. That is, just as 227 is not recalled until + is pressed, 34 is not stored until **RECALL** is pressed.

Short-form addressing can also be used for program addressing and should be used wherever possible to save keystrokes and to make programs more versatile and less dependent upon the partition. (More about this is Section IV.)

Programming Considerations

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 store program instructions from the keyboard only when it is in the learn mode. Conversely, any keystroke (except **STEP, BSTEP, INSRT, DLETE** discussed in *Editing Programs* a few pages later, **CLEAR ALL** and **LEARN**) 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, CLEAR MEM** to position the program pointer at location 0000 and to clear all of program memory.

2. Press **LEARN** 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.

4. Make sure your program does not exceed the program memory size. If too many instructions are entered, the calculator flashes the last location number and the last instruction keyed in.

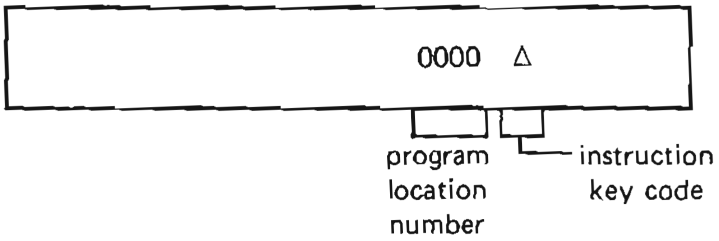
5. Switch from the learn mode to keyboard control by pressing **LEARN.**

6. Run test problems and correct or edit your program according to the procedures outlined in *Editing Programs.*

III 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. Press **CLEAR ALL, NO** and **LEARN** to enter the learn mode. A unique display consisting of a group of zeros with a *null instruction* Δ should appear in the display.

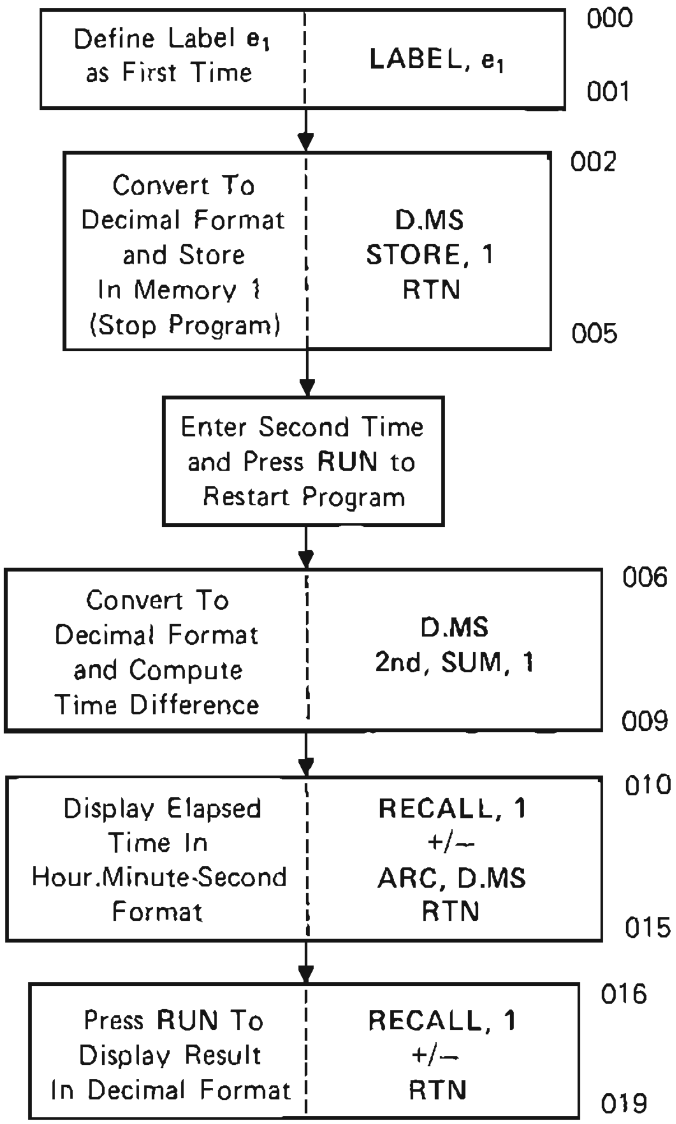


The group of four digits on the left shows you where the program pointer is positioned in program memory. When writing a program, assign each instruction to a location in program memory. This not only allows you to keep track of instructions, but tells the calculator the order in which to complete the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

As each instruction is keyed into the vacant program memory, the location number increments by 1, still displaying a null instruction after the location number. This is because the calculator always displays the *next* location into which an instruction can be placed. You can view the last instruction by pressing the **BSTEP** key.

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 D.MS. See the discussion of this conversion in *Degree Format Conversion* in Section IV.


000
Define Label e_1 as First Time
LABEL, e_1
001
002
Convert To Decimal Format and Store In Memory 1 (Stop Program)
D.MS
STORE, 1
RTN
005
Enter Second Time and Press RUN to Restart Program
006
Convert To Decimal Format and Compute Time Difference
D.MS
2nd, SUM, 1
009
010
Display Elapsed Time In Hour.Minute-Second Format
RECALL, 1
+/−
ARC, D.MS
RTN
015
016
Press RUN To Display Result In Decimal Format
RECALL, 1
+/−
RTN
019

Let's perform this exercise using the following procedure, First, press **2nd, CLEAR MEM** to clear program memory and to place the program pointer at location 0000 and then enter this program according to the procedure outlined below.

Press	Display
I LEARN	0000 △
P LABEL	0001 △
A e_1	0002 △
e_9 D.MS	0003 △
STORE	0004 △
1	0005 △
S RTN	0006 △
e_9 D.MS	0007 △
2nd	0008 △
SUM	0009 △
1	0010 △
RECALL	0011 △
1	0012 △
+/-	0013 △
e_7 ARC	0014 △
e_9 D.MS	0015 △
S RTN	0016 △
RECALL	0017 △
1	0018 △
+/-	0019 △
S RTN	0020 △
I LEARN	0

Since you initially cleared program memory using **2nd, CLEAR MEM**, the key code portion of the display always shows Δ as you enter this program. This is because once a program location is filled the program pointer immediately advances to the next location and displays the key code of the instruction stored there — not the instruction just entered. Note also that short-form addressing has been used here.

Follow this procedure to verify that you have entered the program correctly.

Press	Display
RESET LEARN	0000 Lbl
STEP	0001 e1
STEP	0002 D.MS
STEP	0003 STO
STEP	0004 1
STEP	0005 RTN
STEP	0006 D.MS
STEP	0007 II
STEP	0008 Σ
STEP	0009 1
STEP	0010 RCL
STEP	0011 1
STEP	0012 +/−
STEP	0013 ARC
STEP	0014 D.MS
STEP	0015 RTN
STEP	0016 RCL
STEP	0017 1
STEP	0018 +/−
STEP	0019 RTN
LEARN	0

It is much easier to press **RESET, LIST** to obtain a paper tape listing of the program. **RESET** sends the program pointer to location 0000 just like **QUE**, but **RESET** does not start the program.

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.

Press	Display	Comments
2.15 $\boxed{\text{A}/e_1}$	2.25	t_1 (H.MMSS) → t_1 (H.hh)
3.4254 $\boxed{\text{SPACE/RUN}}$	1.2754	t_2 (H.MMSS) → Δt (H.MMSS)
$\boxed{\text{SPACE/RUN}}$	1.465	→ Δt (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.

Programming Considerations III

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 **STEP**, **BSTEP**, **INSRT**, and **DLETE**. Briefly, **STEP** and **BSTEP** allow you to step forward and backward through program memory and examine its contents one location at a time. From the keyboard, **STEP** may be used to execute a program one step at a time allowing you to observe the results of each operation. The **INSRT** 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 **DLETE** causes the instruction at the current program location to be deleted, shifts all following instructions back one location and fills the last location with a null instruction.

Two additional keys useful in program editing are **GO TO** and **RESET** (Caution: **RESET** resets all flags and clears the subroutine return register. See *Basic Program Control Functions* in Section IV for more). Pressing **RESET** from the keyboard places the program pointer at 0000. Pressing **GO TO** followed by a four-digit absolute program address or a label, repositions the program pointer to that location in program memory. (Leading zeros may be suppressed by short-form addressing.) Pressing **GO TO** 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.

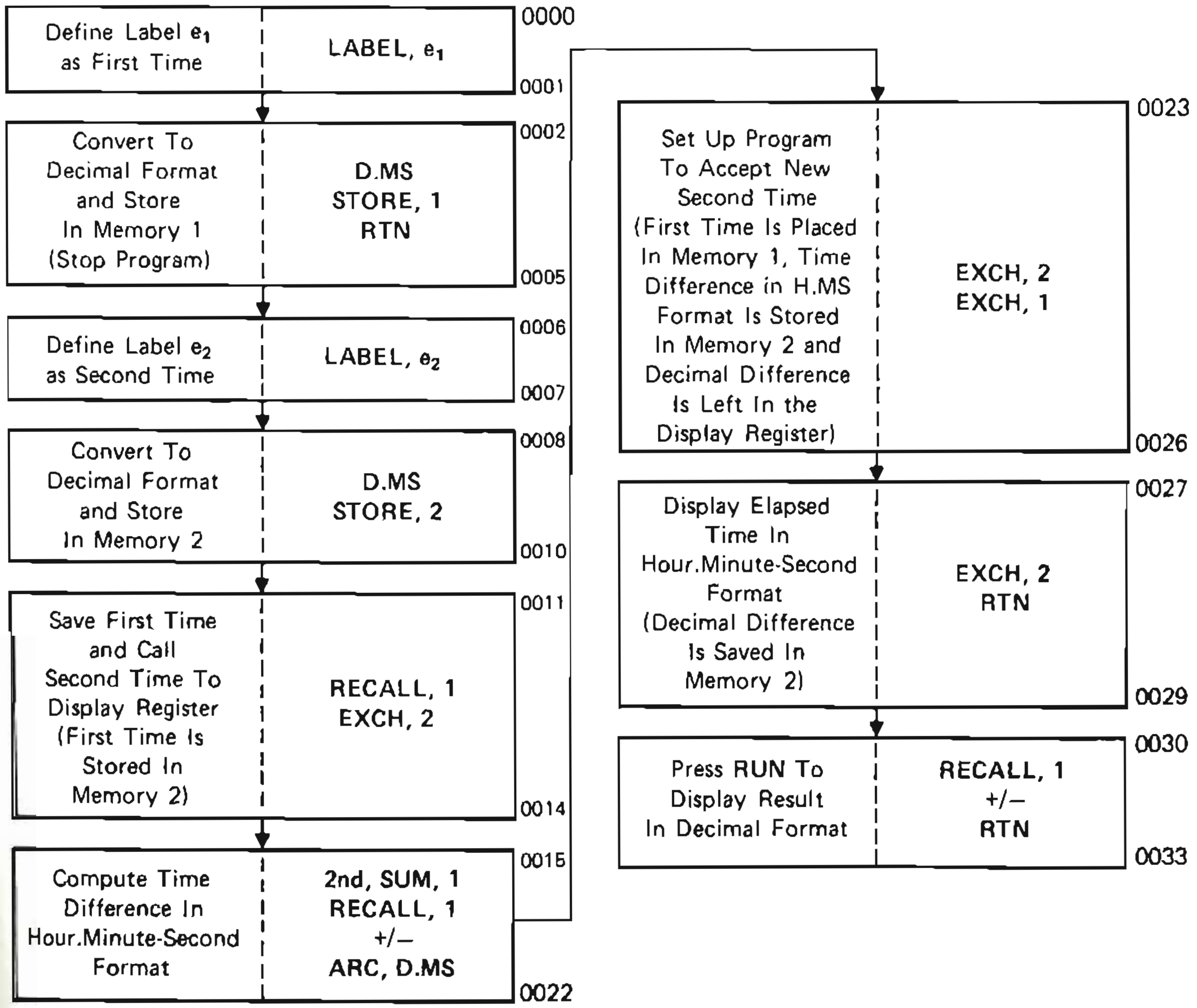
III

Programming Considerations

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.


Define Label e_1 as First Time
LABEL, e_1
0000
0001
Convert To Decimal Format and Store In Memory 1 (Stop Program)
D.MS
STORE, 1
RTN
0002
0005
Define Label e_2 as Second Time
LABEL, e_2
0006
0007
Convert To Decimal Format and Store In Memory 2
D.MS
STORE, 2
0008
0010
Save First Time and Call Second Time To Display Register (First Time Is Stored In Memory 2)
RECALL, 1
EXCH, 2
0011
0014
Compute Time Difference In Hour.Minute-Second Format
2nd, SUM, 1
RECALL, 1
+/−
ARC, D.MS
0015
0022
Set Up Program To Accept New Second Time (First Time Is Placed In Memory 1, Time Difference in H.MS Format Is Stored In Memory 2 and Decimal Difference Is Left In the Display Register)
EXCH, 2
EXCH, 1
0023
0026
Display Elapsed Time In Hour.Minute-Second Format (Decimal Difference Is Saved In Memory 2)
EXCH, 2
RTN
0027
0029
Press RUN To Display Result In Decimal Format
RECALL, 1
+/−
RTN
0030
0033

Make the necessary changes according to the procedure outlined below.

Press	Display	Comments
GO TO 6		Set program pointer to location 0006
LEARN	0006 D.MS	Enter learn mode
INSRT	0006 Δ	Create 2 vacancies
INSRT	0006 Δ	
LABEL	0007 Δ	Insert label e_2
e_2	0008 D.MS	
STEP	0009 II	
INSRT	0009 Δ	Create 2 more vacancies
INSRT	0009 Δ	
STORE	0010 Δ	Insert **STORE 2**
2	0011 2	
INSRT	0011 Δ	Create 4 more vacancies
INSRT	0011 Δ	
INSRT	0011 Δ	
INSRT	0011 Δ	
RECALL	0012 Δ	

Press	Display	Comments
1	0013 Δ	
EXCH	0014 Δ	
2	0015 II	
STEP STEP	0017 1	Single-step past correct instructions
STEP STEP	0019 1	
STEP STEP	0021 ARC	
STEP STEP	0023 RTN	
INSRT INSRT	0023 Δ	Create 2 vacancies
EXCH	0024 Δ	
2	0025 RTN	
INSRT INSRT	0025 Δ	Create 4 vacancies
INSRT INSRT	0025 Δ	
EXCH	0026 Δ	Insert exchanges
1	0027 Δ	
EXCH	0028 Δ	
2	0029 RTN	Exit learn mode
LEARN	0	

Note the use of short-form addressing.

You can now verify that you have modified the program correctly.

Press	Display	Corresponding Keystrokes
RESET LEARN	0000 LBL	LABEL
STEP	0001 e1	e_1
STEP	0002 D.MS	D.MS
STEP	0003 STO	STO
STEP	0004 1	1
STEP	0005 RTN	RTN
STEP	0006 LBL	LABEL
STEP	0007 e2	e_2
STEP	0008 D.MS	D.MS
STEP	0009 STO	STORE
STEP	0010 2	2
STEP	0011 RCL	RECALL
STEP	0012 1	1
STEP	0013 XM	EXCH
STEP	0014 2	2
STEP	0015 II	2nd
STEP	0016 Σ	SUM
STEP	0017 1	1
STEP	0018 RCL	RECALL
STEP	0019 1	1
STEP	0020 +/–	+/–
STEP	0021 ARC	ARC
STEP	0022 D.MS	D.MS
STEP	0023 XM	EXCH
STEP	0024 2	2
STEP	0025 XM	EXCH
STEP	0026 1	1
STEP	0027 XM	EXCH
STEP	0028 2	2
STEP	0029 RTN	RTN
STEP	0030 RCL	RECALL
STEP	0031 1	1
STEP	0032 +/–	+/–
STEP	0033 RTN	RTN
LEARN	0	

Alternately, you can press **RESET, LIST** to list the program onto paper tape.

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

Press	Display	Comments
1.3 A e_1	1.5	t_1 (HH.MMSS) → t_1 (HH.hh)
2.1357 B e_2	0.4357	t_2 (HH.MMSS) → Δt (HH.MMSS)
SPACE RUN	−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 0023-0028 have merely transferred this information to data register 2. Therefore, the problem may be eliminated by changing step 0031. This correction can be made by simply replacing the instruction.

Press	Display
GO TO 3 1	
LEARN	0031 1
2	0032 +/−
LEARN	0

Now, run the program again.

Press	Display	Comments
1.3 A e_1	1.5	t_1 (HH.MMSS) → t_1 (HH.hh)
2.1357 B e_2	0.4357	t_2 (HH.MMSS) → Δt (HH.MMSS)
SPACE RUN	0.7325	→ Δt (HH.hh)
2.1424 B e_2	0.4424	t_2 (HH.MMSS) → Δt (HH.MMSS)
SPACE RUN	0.74	→ Δt (HH.hh)

III 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 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:

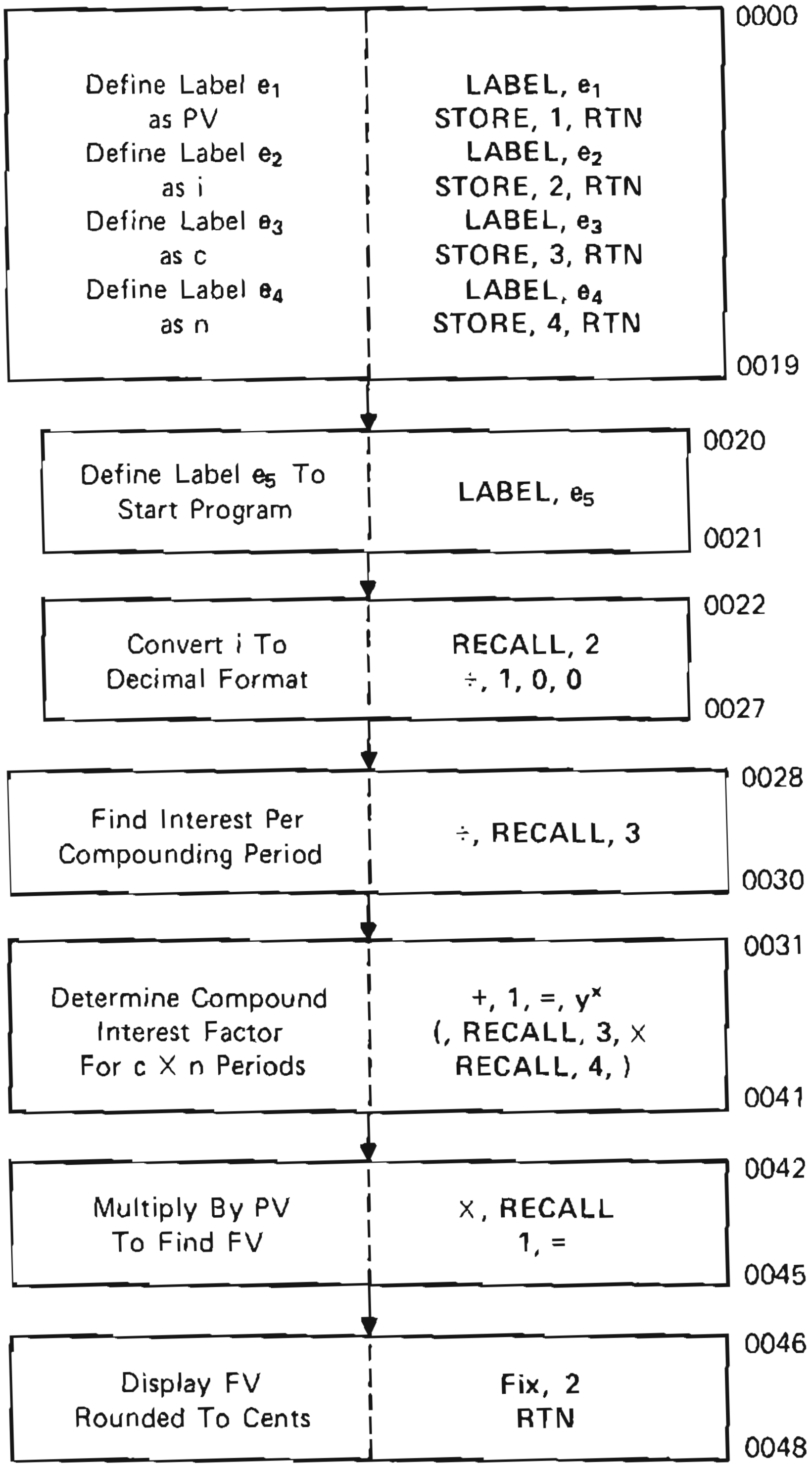
$$FV = PV \times \left[1 + \left(\frac{i}{100} \div c \right) \right]^{cn}$$

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

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.


0000
Define Label e_1 as PV
LABEL, e_1 STORE, 1, RTN
Define Label e_2 as i
LABEL, e_2 STORE, 2, RTN
Define Label e_3 as c
LABEL, e_3 STORE, 3, RTN
Define Label e_4 as n
LABEL, e_4 STORE, 4, RTN
0019
0020
Define Label e_5 To Start Program
LABEL, e_5
0021
0022
Convert i To Decimal Format
RECALL, 2 ÷, 1, 0, 0
0027
0028
Find Interest Per Compounding Period
÷, RECALL, 3
0030
0031
Determine Compound Interest Factor For c X n Periods
+, 1, =, y^x (, RECALL, 3, X RECALL, 4,)
0041
0042
Multiply By PV To Find FV
X, RECALL 1, =
0045
0046
Display FV Rounded To Cents
Fix, 2 RTN
0048

USER INSTRUCTIONS

Step	Procedure	Enter	Press	Display
1	Clear Program Memory and Reset Program Pointer		2nd CLEAR MEM	
2	Enter Learn Mode		Z LEARN	0000 △
3	Enter Investment Calculation Program			
4	Exit Learn Mode		Z LEARN	0
5	Enter Present Value	PV	A e_1	PV
6	Enter Annual Interest	i	B e_2	i
7	Enter Number of Compounding Periods Per Year	c	C e_3	c
8	Enter Number of Years	n	D e_4	n
9	Compute Future Value		E e_5	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
0000 LBL	P LABEL
0001 e1	A e1
0002 STO	STORE
0003 1	1
0004 RTN	$ RTN
0005 LBL	P LABEL
0006 e2	B e2
0007 STO	STORE
0008 2	2
0009 RTN	$ RTN
0010 LBL	P LABEL
0011 e3	C e3
0012 STO	STORE
0013 3	3
0014 RTN	$ RTN
0015 LBL	P LABEL
0016 e4	D e4
0017 STO	STORE
0018 4	4
0019 RTN	$ RTN
0020 LBL	P LABEL
0021 e5	E e5
0022 RCL	RECALL
0023 2	2
0024 ÷	÷
0025 1	1
0026 0	0
0027 0	0
0028 ÷	÷
0029 RCL	RECALL
0030 3	3
0031 +	+
0032 1	1
0033 =	=
0034 y^x	y^x
0035 (	(
0036 RCL	RECALL
0037 3	3
0038 X	×
0039 RCL	RECALL
0040 4	4
0041)	)
0042 X	×
0043 RCL	RECALL
0044 1	1
0045 =	=
0046 FIX	FIX
0047 2	2
0048 RTN	$ RTN

Investment Calculation Program

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

Enter	Press	Display	Comments
3000	A e_1	3000.	PV
8	B e_2	8.	i
365	C e_3	365.	c
5	D e_4	5.	n
	E e_5	4475.28	FV
12	C e_3	12.00	c
	E e_5	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, subtract 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 quantities 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	

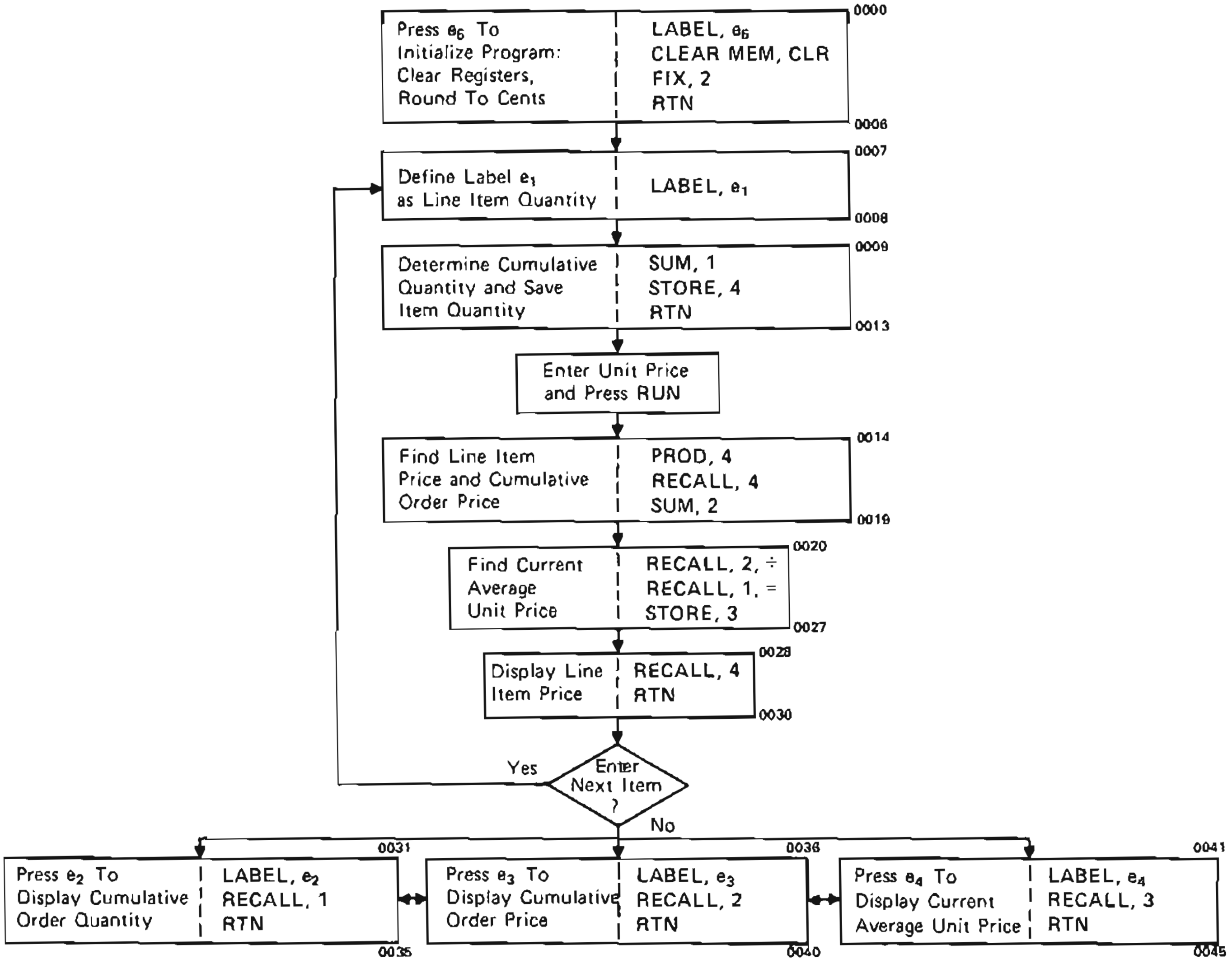
A glance at the order immediately tells one to multiply, add, and divide. The key is how to organize the problem and what to instruct the calculator to do.

First, decide how to enter your variables. In the following example, 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. Prestoring the values would have limited the capability of the program by fixing the number of line items that could be handled for any one order. Using memory arithmetic to assimilate the values as they are entered allows an order to have an unlimited number of line items.

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


0000
Press e_6 To Initialize Program: Clear Registers, Round To Cents
LABEL, e_6
CLEAR MEM, CLR
FIX, 2
RTN
0006
0007
Define Label e_1 as Line Item Quantity
LABEL, e_1
0008
0009
Determine Cumulative Quantity and Save Item Quantity
SUM, 1
STORE, 4
RTN
0013
Enter Unit Price and Press RUN
0014
Find Line Item Price and Cumulative Order Price
PROD, 4
RECALL, 4
SUM, 2
0019
0020
Find Current Average Unit Price
RECALL, 2, ÷
RECALL, 1, =
STORE, 3
0027
0028
Display Line Item Price
RECALL, 4
RTN
0030
Yes
Enter Next Item ?
No
0031
Press e_2 To Display Cumulative Order Quantity
LABEL, e_2
RECALL, 1
RTN
0035
0036
Press e_3 To Display Cumulative Order Price
LABEL, e_3
RECALL, 2
RTN
0040
0041
Press e_4 To Display Current Average Unit Price
LABEL, e_4
RECALL, 3
RTN
0045

Pricing Control Program

USER INSTRUCTIONS

Step	Procedure	Enter	Press	Display
1	Clear Program Memory and Reset Program Pointer		2nd CLEAR MEM	
2	Enter Learn Mode		Z LEARN	0000 Δ
3	Enter Pricing Control Program			
4	Exit Learn Mode		Z LEARN	0
5	Initialize Program		E e_5	0.00
6	Enter Line Item Quantity	Quantity	A e_1	Quantity
7	Enter Unit Price	Unit Price	SPACE RUN	Line Item Price
	Repeat Steps 6 and 7 for Each Line Item			
	After Each Line Item Entry the Following Variables May Be Displayed:			
	Cumulative Quantity		B e_2	Order Quantity
	Cumulative Cost		C e_3	Order Price
	Average Unit Price		D e_4	Average Price

Location and Key Code	Key Sequence
0000 LBL	P LABEL
0001 e_5	E e_5
0002 CMS	CLEAR MEM
0003 CLR	CLEAR
0004 FIX	FIX
0005 2	2
0006 RTN	S RTN
0007 LBL	P LABEL
0008 e_1	A e_1
0009 SUM	SUM
0010 1	1
0011 STO	STORE
0012 4	4
0013 RTN	S RTN
0014 Π	PROD
0015 4	4
0016 RCL	RECALL
0017 4	4
0018 SUM	SUM
0019 2	2
0020 RCL	RECALL
0021 2	2
0022 ÷	÷
0023 RCL	RECALL
0024 1	1
0025 =	=
0026 STO	STORE
0027 3	3
0028 RCL	RECALL
0029 4	4
0030 RTN	S RTN
0031 LBL	P LABEL
0032 e_2	B e_2
0033 RCL	RECALL
0034 1	1
0035 RTN	S RTN
0036 LBL	P LABEL
0037 e_3	C e_3
0038 RCL	RECALL
0039 2	2
0040 RTN	S RTN
0041 LBL	P LABEL
0042 e_4	D e_4
0043 RCL	RECALL
0044 3	3
0045 RTN	S RTN

Pricing Control Program

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

Press	Display	Comments
E e_5	0.00	Initialize
100 A e_1	100.00	Quantity A
.25 SPACE RUN	25.00	Unit Price A Line Item Price
200 A e_1	200.00	Quantity B
.15 SPACE RUN	30.00	Unit Price B Line Item Price
50 A e_1	50.00	Quantity C
.35 SPACE RUN	17.50	Unit Price C Line Item Price
150 A e_1	150.00	Quantity D
.4 SPACE RUN	60.00	Unit Price D Line Item Price
300 A e_1	300.00	Quantity E
.1 SPACE RUN	30.00	Unit Price E Line Item Price
B e_2	800.00	Total Order Quantity
C e_3	162.50	Total Order Price
D e_4	0.20	Avg. Unit Price (Rounded)
FIX 9	0.203125	Avg. Unit Price (Exact)

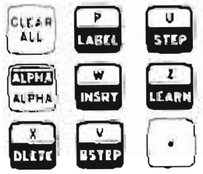
Programming Considerations III

ADVANCED PROGRAMMING

MORE ABOUT LABELS

As you remember, the user-defined keys ($e_1 - e_5$, $e_6 - e_{15}$) 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 key on the calculator as a label. For instance, **2nd**, **sin**, **x^2**, **=**, CLEAR, EE, FIX and others can be labels. These are called *Program Labels.* Only the following keys cannot be used as labels.



(if secondary labels are used)

Numbers (0–9)

The only difference between the program labels and the user-defined labels is that pressing a program 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 GO TO, x^2, RUN does cause the program to start running at label x^2. Nonetheless, you now have over 60 more labels to work with.

Program labels can be used anywhere in a program as can the user-defined labels. Throughout the remainder of this section we'll primarily discuss user-defined keys as labels because of their versatility.

Occasionally you may need more labels than the 77 offered here. See Section IV, *Functions of a Label,* for more information.

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 to bottom" 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.

III Programming Considerations

UNCONDITIONAL TRANSFERS

RESET, **GO TO** and **SUBR** are called unconditional transfer instructions. **RESET** automatically positions the program pointer at location 0000. **GO TO** and **SUBR** unconditionally place the pointer at the location you specify. Note also that **RESET** clears the subroutine return register and resets all program flags.

The Go To Instruction — GO TO

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, **GO TO** followed by an absolute address (program memory addresses) 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. In the learn mode, for example, pressing **GO TO, 9** is the same as **GO TO, 0009** if and only if the next keystroke is not a number. Observe the following little counting program.

Press	Display	Comments
CLEAR 2nd CLEAR MEM	0	Prepare for program
GO TO 9 LEARN	0009 Δ	Go To location 0009 and enter learn mode
+	0010 Δ	Key in program
1	0011 Δ	
=	0012 Δ	
PAUSE	0013 Δ	
GO TO	0014 Δ	
0	0015 Δ	
0	0016 Δ	
0	0017 Δ	
9	0018 Δ	
LEARN	0	Exit learn mode

Now, press **GO TO, 9, RUN** and watch the calculator count.

Try the following exercise on your calculator.

Press	Display	Comments
2nd CLEAR MEM CLEAR	0	Clear program memory and display
GO TO 136	0	Sends the program pointer to location 0136
LEARN	0136 Δ	
RTN	0137 Δ	Store RTN in location 0136
LEARN RESET LEARN	0000 Δ	Return to 0000
GO TO	0001 Δ	Store GO TO in 0000
0	0002 Δ	
1	0003 Δ	
3	0004 Δ	
6	0005 Δ	
LEARN	0	Exit learn mode
RESET RUN	0	Execute program
LEARN	0137 Δ	Transfer is instantly made to location 0136 where the RTN stored there is executed, leaving the program pointer at location 0137.

The same thing would work for a label. Pressing GO TO, 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.

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.

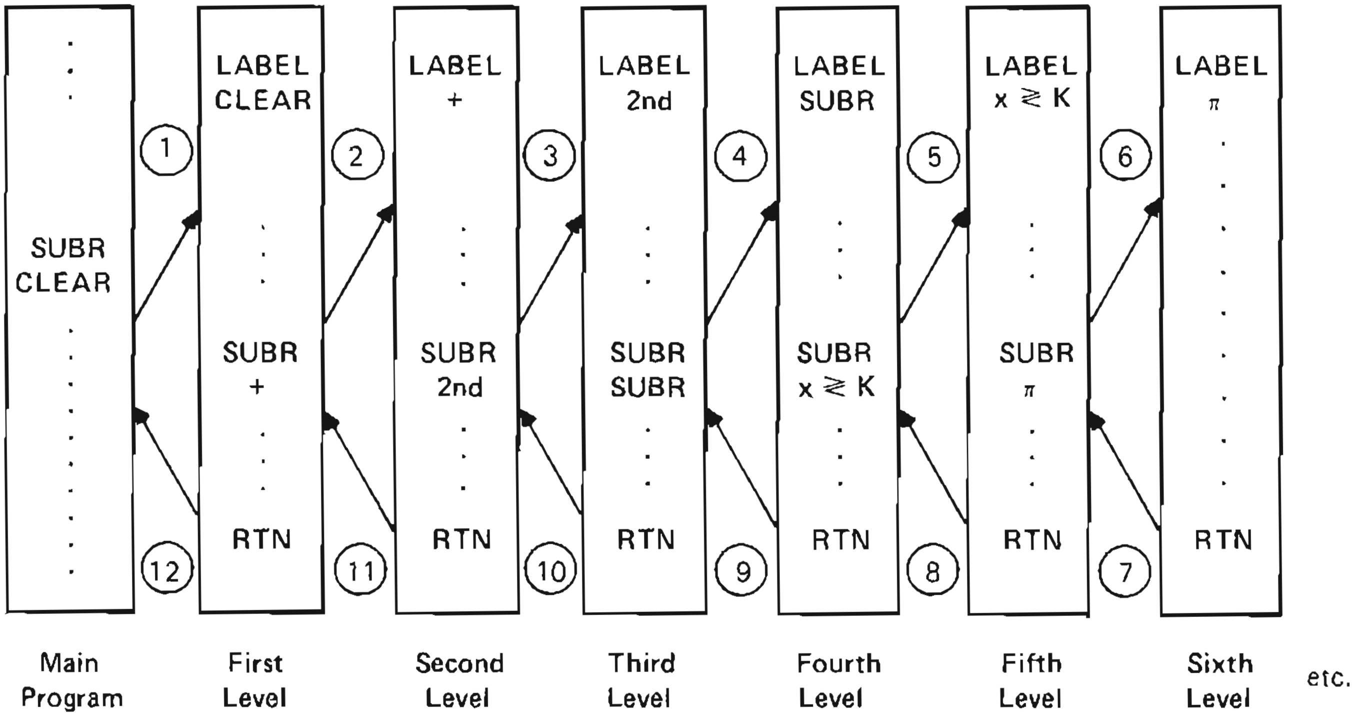
It's a good programming practice to write your programs so that they can be used as subroutines. Now, they can be used by other programs without having to be modified. You may do this by simply using RTN to stop program execution instead of **HALT**. The remaining programs in this section are written using this technique.

The Subroutine Instruction – SUBR

The subroutine instruction is a Go To that has been modified so that when used in a program, it remembers where it transferred from. From the keyboard, pressing **SUBR 0136** sends the program pointer instantly to location 0136, just like the GO TO instruction.

If **SUBR 0136** had been keyed into locations 0000–0004 as **GO TO 0136** was in the previous example, it would have been executed the same way. But, when executed, program location 0005 is stored in the *Subroutine Return Register*. Now, if there is a calculation sequence beginning at 0136 that you have ended with **RTN**, processing would look at the subroutine return register, find 0005, and bounce back to that location where processing continues. To get processing to go back where it came from, simply end the subroutine with **RTN**. What actually has happened is that you have transferred to a subroutine at 0136 and return is automatic when the RTN instruction is encountered.

Actually, as many as twelve 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 twelve times! Part of 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 RTN 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.

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 subroutine instruction built in. So, if you label a program part with a user-defined key and end it with RTN, that part is processed just as though you had called it with the SUBR key.

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 logically end with **RTN** to instruct processing to return to the sequence that called it.

There are three methods of calling subroutines.

- Absolute address, **SUBR 0136**
- Common label, **SUBR, x^2**
- User-defined label, **e_1**

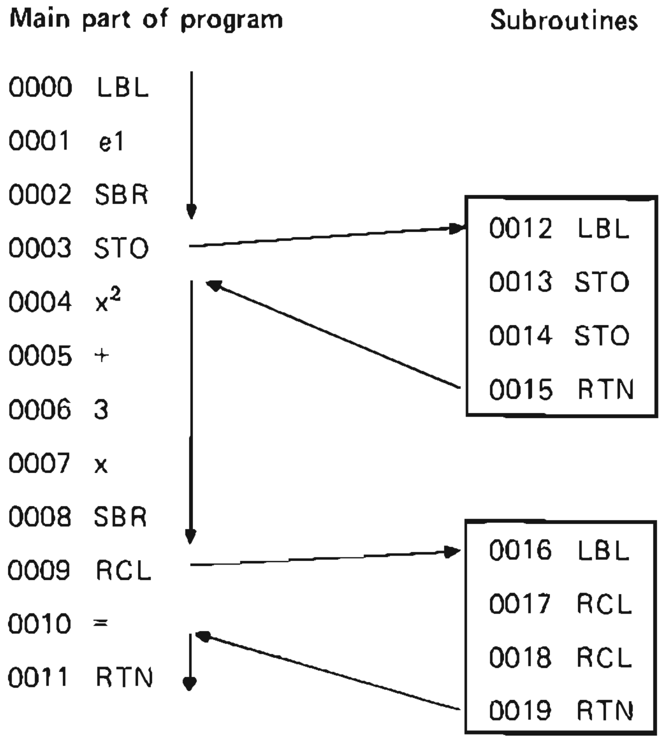
Note that user-defined labels do not need to be preceded by **SUBR.**

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 immediately regardless of where it is. This is because a table of labels and their locations is built each time you exit the learn mode. 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.

```
STORE
1
x²
+
3
×
RECALL
1
=
5
RTN
```

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



```
Main part of program        Subroutines

0000  LBL
0001  e1
0002  SBR
0003  STO                   0012  LBL
0004  x²                    0013  STO
0005  +                     0014  STO
0006  3                     0015  RTN
0007  x
0008  SBR
0009  RCL                   0016  LBL
0010  =                     0017  RCL
0011  RTN                   0018  RCL
                            0019  RTN
```

Enter any x value and press e_1. 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. Short-form addressing has been used to store and recall values from data register 0.

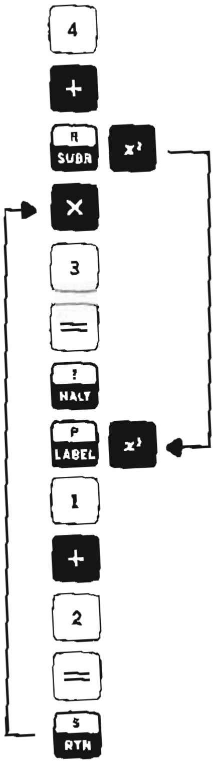
Things to Watch Out For in Subroutines

Two instructions that should be used very carefully in subroutines are Reset 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 0000 (the primary function of **RESET**) in a subroutine, use **GO TO 0000** or a label if there is one at 0000.

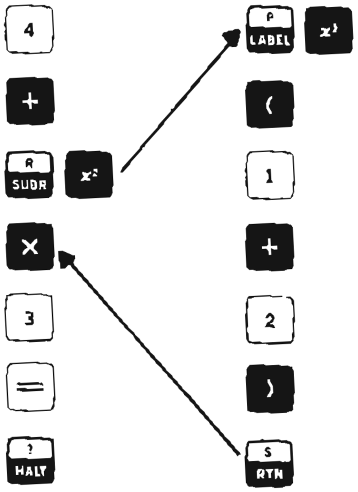
The equals instruction completes *all* pending operations. If used in a subroutine, the pending operations not only of the subroutine, but those of the main program are completed as well.

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



The equals here in subroutine x^2 not only completes 1 + 2, but also the 4 + before returning to the X, 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.



This sequence yields the correct answer, 13.

Beginning each subroutine with (and placing) just before the **RTN** is a good habit to develop. It takes an extra keystroke as opposed to using equals, 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 reuse 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 × (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 or clear the return register using **2nd CLEAR MEM, CLEAR ALL** or **RESET**, 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 **RESET** instruction to clear the subroutine return register. You may do this manually, but it is preferable, whenever possible, to include **RESET** at a proper point in the program.

CONDITIONAL TRANSFERS (DECISION MAKERS)

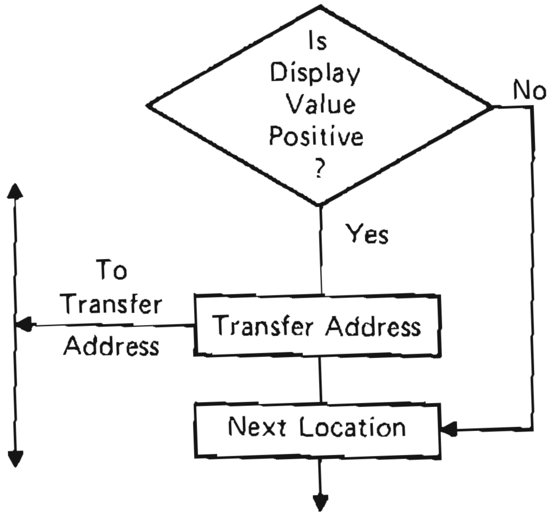
Other features that are very useful in problem solving, are instructions that are capable of making decisions in your programs. A family of what are called *conditional transfer instructions* make this decision-making process possible. Each time one of them is encountered in a program, it makes some 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. The value in the display register: positive, negative, zero or invalid due to an error condition.
2. Contents of data register 0.
3. Status of program flags.

A transfer address follows each of these branching instructions. When the answer to any test is "yes" (test true), transfer is made to that address. If the answer to test is "no" (test false), the transfer is skipped. For instance, if the test **IF POS, e_1** is positive, transfer is made to e_1 just as if **GO TO, e_1** had been encountered. If the tests are not exactly true, no transfer takes place.

There are several instructions available to analyze the current display value at any point in a program – **IF POS** if positive, **IF ZRO** if zero, and **IF ERR** if error. These instructions test the display value to see if it is positive, negative, zero or in error and transfer accordingly. An address, either absolute or label, must follow each of these instructions whenever they are used. When a test is made, for instance, "Is the display value positive?" and the answer is "yes," a transfer is made to the address following the transfer instruction. If not, the address is ignored and processing continues as though no test had ever been made. Graphically, here's what happens.



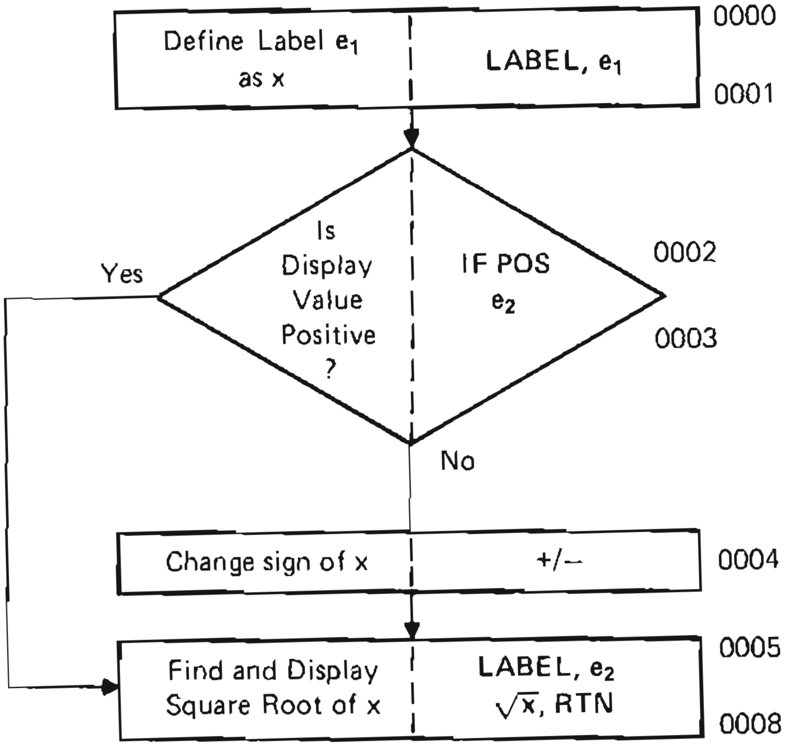
These instructions are designed for use within a program to direct the flow of processing, but they can be used from the keyboard as well. Try these few keystrokes:

Press	Display	Comments
5	5	Place 5 in the display
IF POS 1111	5	If this display value is positive, go to location 1111
LEARN	1111 Δ	Enter learn mode to verify transfer
LEARN	0	Exit learn mode
5 +/-	–5	Place –5 in display
IF POS 1234	–5	If positive, go to location 1234
LEARN	1111 Δ	No transfer took place because –5 is not positive.

This and the other display value test instructions work the same way when encountered in a program.

Square Root Example

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



Location and Key Code	Key Sequence
0000 LBL	P LABEL
0001 e1	A e_1
0002 IF +	N IF POS
0003 e2	B e_2
0004 +/–	+/–
0005 LBL	P LABEL
0006 e2	B e_2
0007 $\sqrt{x}$	$\sqrt{x}$
0008 RTN	S RTN

To exercise the program, enter any number and press e_1. Enter 4 e_1 and see 2. Enter −4 e_1 and again see 2.

These instructions can also be used with the 2nd key to reverse their tests and transfer accordingly as shown below.

Instruction Sequence	Question Asked (Test Made)
IF ERR	Does an error condition exist?
2nd IF ERR	Does an error condition not exist?
IF POS	Is the display value positive or zero?
2nd IF POS	Is the display value negative?
IF ZRO	Is the display value zero?
2nd IF ZRO	Is the display value nonzero?

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.

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 z, press **S FLG** z
- To reset flag z, press **2nd**, **S FLG** z
- To test flag z and transfer if it is set, press **T FLG** z then complete the instruction with a transfer address just like the test instructions, mentioned earlier.
- To test flag z and transfer if it is not set, press **2nd**, T **FLG** z 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.

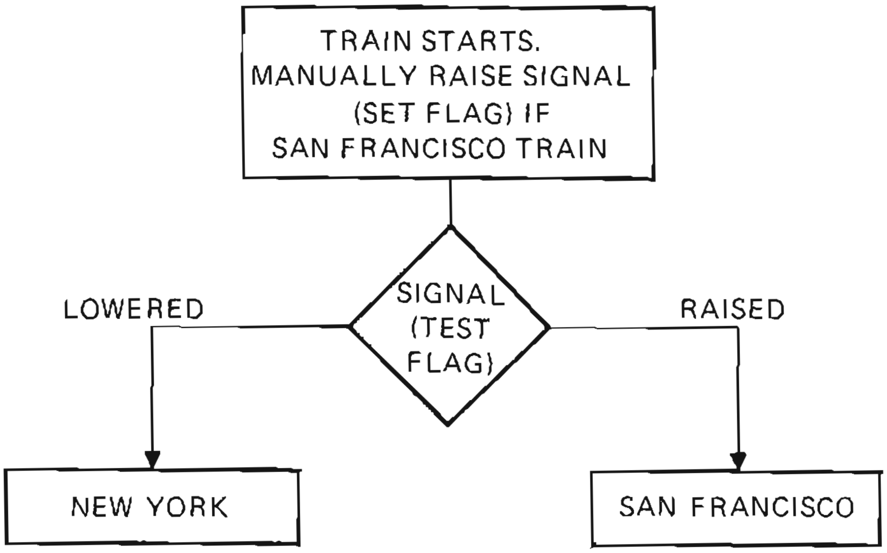
Press	Display	Comments
2nd CLEAR MEM	0	Clear program memory. This instruction also resets or turns off all program flags.
S FLG 4	0	Set flag number 4
T FLG 4 1360	0	Test flag 4, if set, go to location 1360
LEARN	1360 Δ	Transferred to 1360 because flag is set
LEARN T FLG 5 1111	0	Test flag 5, if set, go to location 1111
LEARN	1360 Δ	Did not transfer because flag is not set
LEARN 2nd S FLG 4	0	Reset flag 4
T FLG 4 2	0	Test flag 4, if set, go to location 0002
LEARN	1360 Δ	Flag is not set, so no transfer is made
LEARN 2nd T FLG 4 2	0	Test "If flag is not set – transfer to 0002"
LEARN	0002 Δ	Flag not set, so transfer to 0002 is performed.

The Flag test instruction behaves similar to the display value tests. The difference is that these instructions test flags instead of the display value. 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 is not set, and the testing of a flag have no effect on the status of the flag, nor do they affect calculations. All 10 flags can be reset at once with RESET, 2nd CLEAR MEM or CLEAR ALL.

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

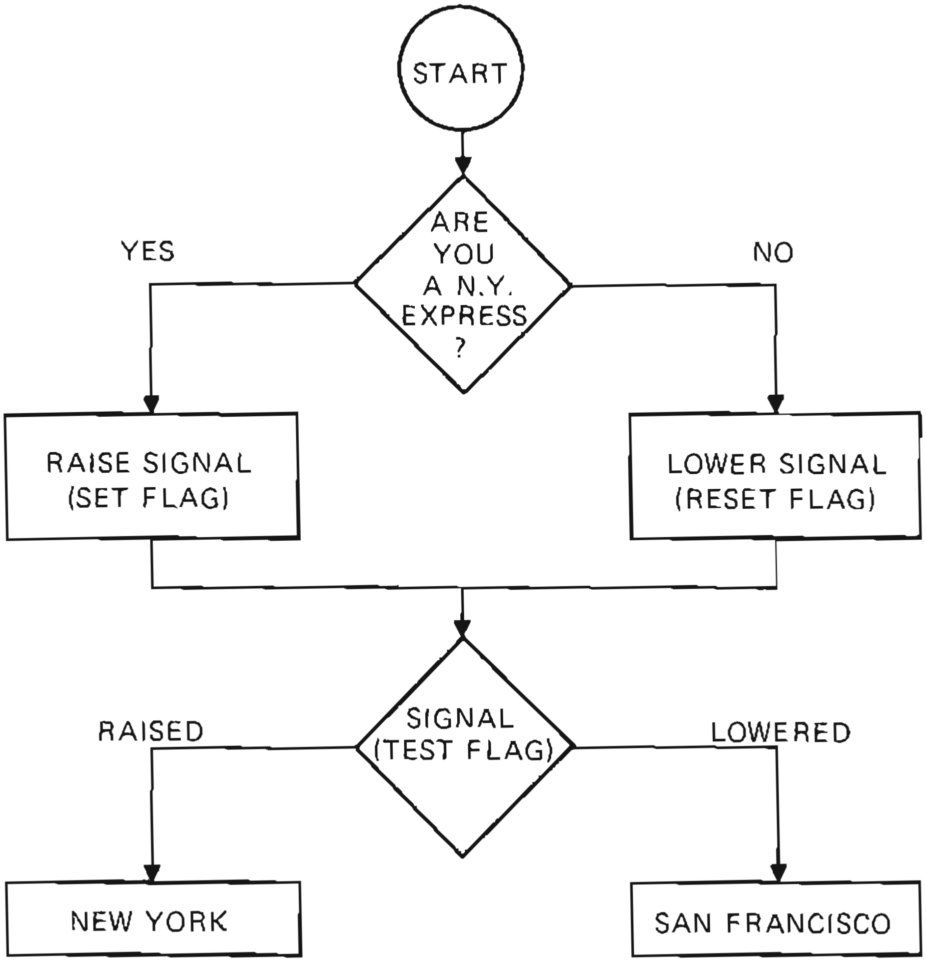
The following situation illustrates the first of the three uses mentioned for flags, how they can be used manually from the keyboard. Suppose you are a train dispatcher in a switch tower. A train is going down the track. It encounters a signal at a junction. If the signal is raised, the train is switched 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 other than 0?" or many other questions. If the answer is "yes," set a flag and test it later when you need to choose processing options.

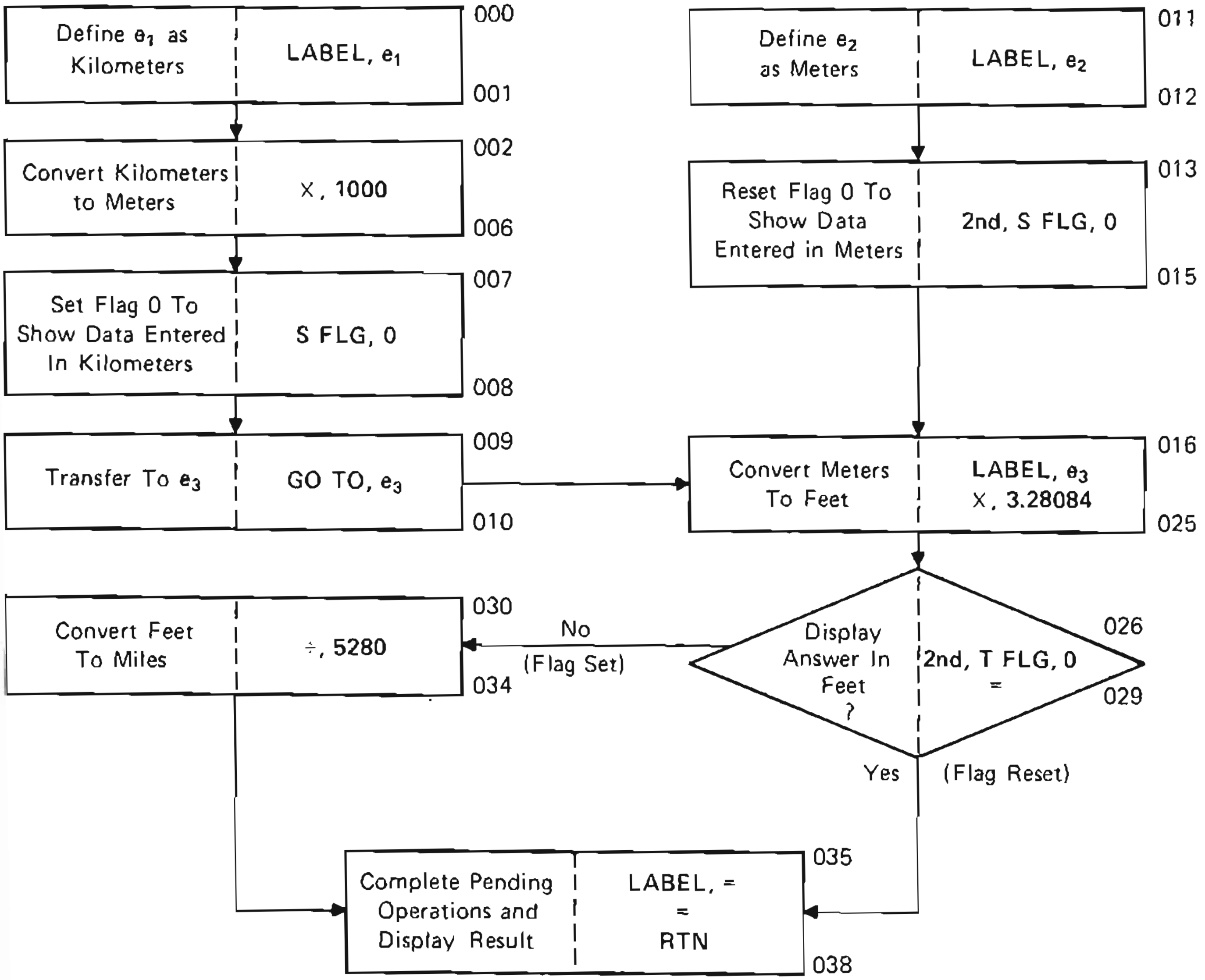
The third use of flags gives the program a means of remembering how it reached a given point. This is necessary in situations where what you wish to do depends on which path your program has taken. You recall that the program pointer only knows where it is and has no recollection of how it got there. Such recollective ability is sometimes needed, however, and the program flags provide a convenient way to do this. Just place **S FLG** z in one path and **2nd**, **S FLG** z 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 **T FLG** z. 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.

Three of the ten flags (7, 8 and 9) possess special program controlling features in addition to their normal functions. See *Flag Operation* in Section IV before using these flags.

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] [CLEAR MEM]	
2	Enter Learn Mode		[Z LEARN]	0000 Δ
3	Enter Metric Conversion Program			
4	Exit Learn Mode		[Z LEARN]	0
5	Enter Kilometers OR	Kilometers	[A e_1]	Miles
	Enter Meters and Compute Result	Meters	[B e_2]	Feet

Location and Key Code		Key Sequence	Location and Key Code		Key Sequence
0000	LBL	P LABEL	0020	•	·
0001	e1	A e_1	0021	2	2
0002	X	×	0022	8	8
0003	1	1	0023	0	0
0004	0	0	0024	8	8
0005	0	0	0025	4	4
0006	0	0	0026	II	2nd
0007	SF	I S FLG	0027	TFS	J T FLG
0008	0	0	0028	0	0
0009	GTO	$ GO TO	0029	=	=
0010	e3	C e_3	0030	÷	÷
0011	LBL	P LABEL	0031	5	5
0012	e2	B e_2	0032	2	2
0013	II	2nd	0033	8	8
0014	SF	I S FLG	0034	0	0
0015	0	0	0035	LBL	P LABEL
0016	LBL	P LABEL	0036	=	=
0017	e3	C e_3	0037	=	=
0018	X	×	0038	RTN	5 RTN
0019	3	3			

Metric Conversion Program

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

Enter	Press	Display	Comments
50	B e_2	164.042	Meters → Feet
90	A e_1	55.92340909	Kilometers → Miles

Caution: short-form addressing cannot be used for flag 0.

Data Register Transfers — Decrement and Skip on Zero

This powerful instruction sequence uses the contents of data register 0 to decide whether or not to transfer. This technique, called *Decrement and Skip on Zero (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 sends the program pointer to an earlier location.

Unconditional Looping

There are two methods of unconditional looping.

RESET loops back to program location 0000

GO TO loops back to wherever you tell it.

Let's create a program to count by fours. The simple sequence +, 4, =, PAUSE, RESET 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 0000, enter a starting number and press RUN and watch it count. If you were to place the sequence in program memory starting at location 0020, you could replace **RESET** with **GO TO 0020** and accomplish the same thing. Just remember that initially you have to begin execution at location 0020.

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

To exit from a loop, place a conditional 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
0000 RCL	RECALL	Recall previous result from R_0
0001 +	+	
0002 4	4	
0003 =	=	
0004 PAU	Q PAUSE	
0005 STO	STORE	Store result in R_0
0006 –	–	
0007 2	2	
0008 0	0	
0009 =	=	Subtract 20 from result
0010 IFZ	O IF ZRO	
0011 HLT	T HALT	
0012 RST	Y RESET	
0013 LBL	P LABEL	
0014 HLT	T HALT	
0015 RTN	S RTN	

Once this program is stored in program memory, just press **CLEAR MEM, QUE.** The conditional test in location 0010 tests each count –20 and does nothing until the count reaches 20 (count –20 reaches 0) when it transfers to Label HALT and stops. The looping is handled here by **RESET.**

Conditional Looping

The counting example can also be totally controlled by a conditional transfer instruction. Again, let's count from 0 to 20.

Location and Key Code	Key Sequence	Comments
0000 CMS	CLEAR MEM	Clear data memories
0001 LBL	P LABEL	
0002 e1	A e_1	
0003 RCL	RECALL	Recall previous result from R_0
0004 +	+	
0005 4	4	
0006 =	=	
0007 PAU	Q PAUSE	
0008 STO	STORE	Store result in R_0
0009 –	–	
0010 2	2	
0011 0	0	
0012 =	=	Subtract 20 from result
0013 II	2nd	
0014 IF +	H IF POS	If negative, go to e_1
0015 e1	A e_1	
0016 RTN	S RTN	Halt when result becomes 0

Here, **2nd**, **IF POS** controls the looping.

To run the program, simply press QUE.

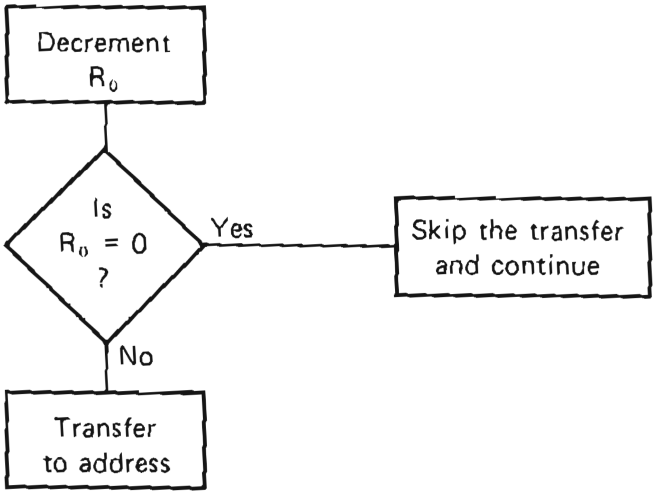
III

Programming Considerations

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 for this technique is **2nd, GO TO** followed by a transfer address.

This versatile transfer decreases the magnitude of the contents of data register 0 by 1 (if the data register contents are less than 1, they are decremented to 0), then tests the contents of register 0. (For this discussion, let R_0 represent the contents of data register 0.) If R_0 is zero, the transfer address is skipped. Otherwise, the address causes the processing sequence to transfer. DSZ decrements register 0 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 CLEAR MEM	0	Clear program memory
2 STORE	2.	Store 2 in data register 0
2nd GO TO 1360	2.	Decrements R_0 by 1, then asks, "Is R_0 = 0?" If no, transfer
LEARN	1360 Δ	Transfer made to 1360
LEARN RECALL 00	1.	R_0 was 2 and now 1 because of DSZ
2nd GO TO 1111	1.	Decrement and test again
LEARN	1360 Δ	No transfer because R_0 = 0 now
LEARN RECALL 00	0.	R_0 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 fours 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
0000	CMS	CLEAR MEM	Clear all data memories
0001	5	5	
0002	XM	EXCH	Store 5 in register 0 and clear display
0003	LBL	P LABEL	Label this part e1
0004	e1	A e_1	
0005	+	+	
0006	4	4	
0007	=	=	
0008	PAU	Q PAUSE	Display each count
0009	II	2nd	Decrement register 0 and test to see if
0010	GTO	$ GO TO	R_0 is less than 1
0011	e1	A e_1	If R_0 is greater than 0, transfer to e1
0012	RTN	S RTN	Stops when R_0 is zero

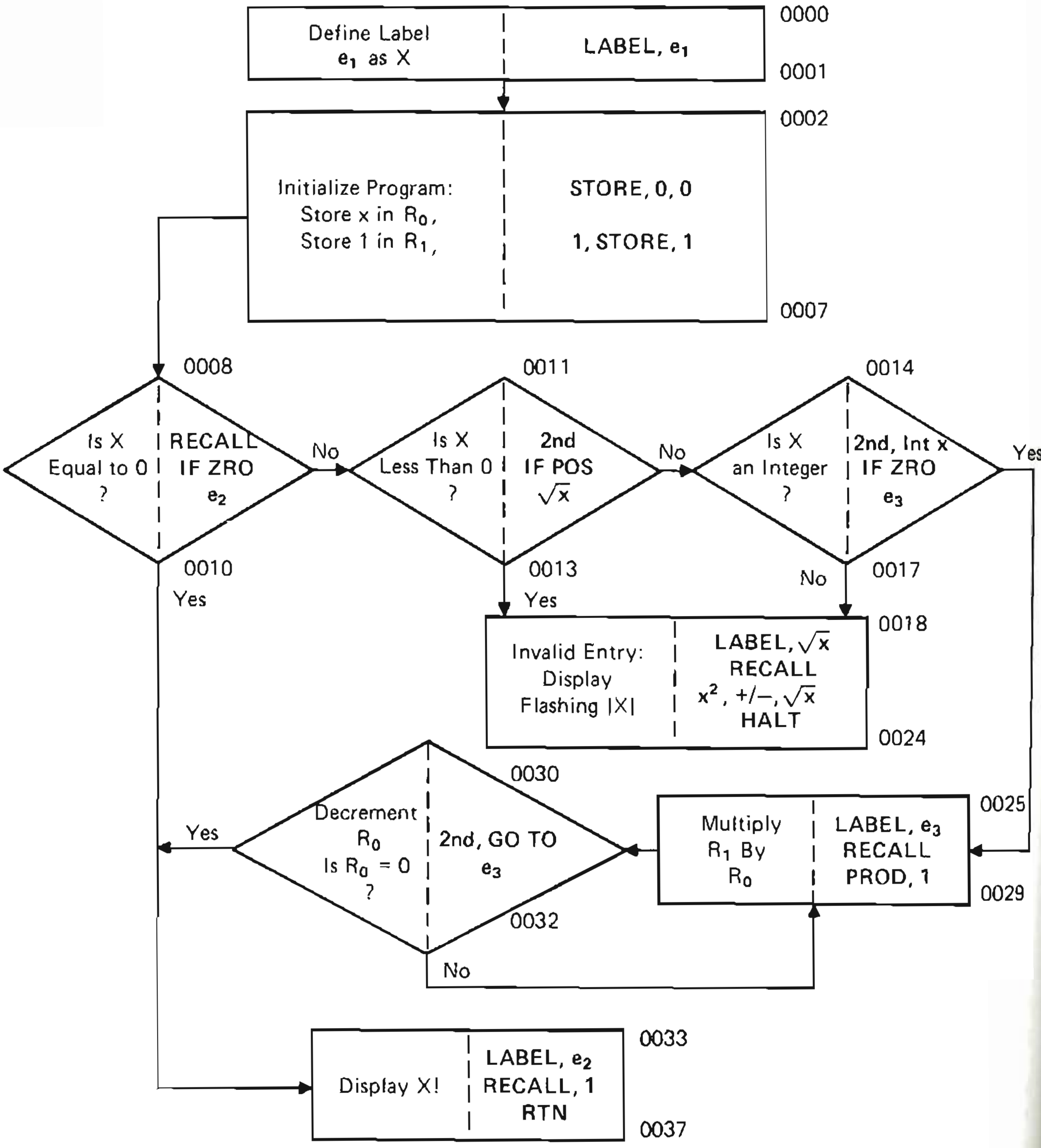
DSZ can increment R_0 (add 1 to R_0) from the negative side of zero as well. A –5 could have been used just as well in the above example.

For more details, see *Decrement and Skip on Zero* in Section IV.

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 · (x – 1) · (x – 2) · . . . . · 2 · 1. By definition of this function, X must be a positive integer and 0! = 1.


0000
Define Label
e_1 as X
LABEL, e_1
0001
0002
Initialize Program:
Store x in R_0,
Store 1 in R_1,
STORE, 0, 0
1, STORE, 1
0007
0008
Is X
Equal to 0
?
RECALL
IF ZRO
e_2
0010
Yes
No
0011
Is X
Less Than 0
?
2nd
IF POS
$\sqrt{x}$
0013
Yes
No
0014
Is X
an Integer
?
2nd, Int x
IF ZRO
e_3
0017
No
Yes
0018
Invalid Entry:
Display
Flashing |X|
LABEL, $\sqrt{x}$
RECALL
x^2, +/−, $\sqrt{x}$
HALT
0024
0030
Decrement
R_0
Is R_0 = 0
?
2nd, GO TO
e_3
0032
Yes
No
0025
Multiply
R_1 By
R_0
LABEL, e_3
RECALL
PROD, 1
0029
0033
Display X!
LABEL, e_2
RECALL, 1
RTN
0037

X! Program

Location and Key Code	Key Sequence	Location and Key Code	Key Sequence
0000 LBL	P LABEL	0019 $\sqrt{x}$	$\sqrt{x}$
0001 e1	A e₁	0020 RCL	RECALL
0002 STO	STORE	0021 x^2	x^2
0003 0	0	0022 +/–	+/–
0004 0	0	0023 $\sqrt{x}$	$\sqrt{x}$
0005 1	1	0024 HLT	I HALT
0006 STO	STORE	0025 LBL	P LABEL
0007 1	1	0026 e3	C e₃
0008 RCL	RECALL	0027 RCL	RECALL
0009 IFZ	O IF ZRO	0028 Π	PROD
0010 e2	B e₂	0029 1	1
0011 II	2nd	0030 II	2nd
0012 IF +	H IF POS	0031 GTO	$ GO TO
0013 $\sqrt{x}$	$\sqrt{x}$	0032 e3	C e₃
0014 II	2nd	0033 LBL	P LABEL
0015 INT	Int x	0034 e2	B e₂
0016 IFZ	O IF ZRO	0035 RCL	RECALL
0017 e3	C e₃	0036 1	1
0018 LBL	P LABEL	0037 RTN	S RTN

XI Program

In the sample program, 1 is stored in R_1 to allow multiplication by memory arithmetic and for the definition of 0!. 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 deliberately created at location 0023 halts the program and flashes the absolute value of the number entered. The actual calculation loop occurs between locations 0027–0031.

Use of this program is very straightforward. Simply enter an X value less than 70 and press e_1. (70! overflows the calculation limits of the calculator.)

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

Press	Display	Comments
6 A/e_1	720	6!
2 +/– A/e_1	"2 ?"	Invalid entry
CLEAR	0	Clear error
0 A/e_1	1	0! = 1 by definition
7.3 A/e_1	"7.3 ?"	Invalid entry
CLEAR	0	Clear error
39 A/e_1	2.039788208 46	39!

Note: Quote markes in the display column indicate a flashing display.

III Programming Considerations

MORE ON APPLICATIONS

BOND COST PROGRAM

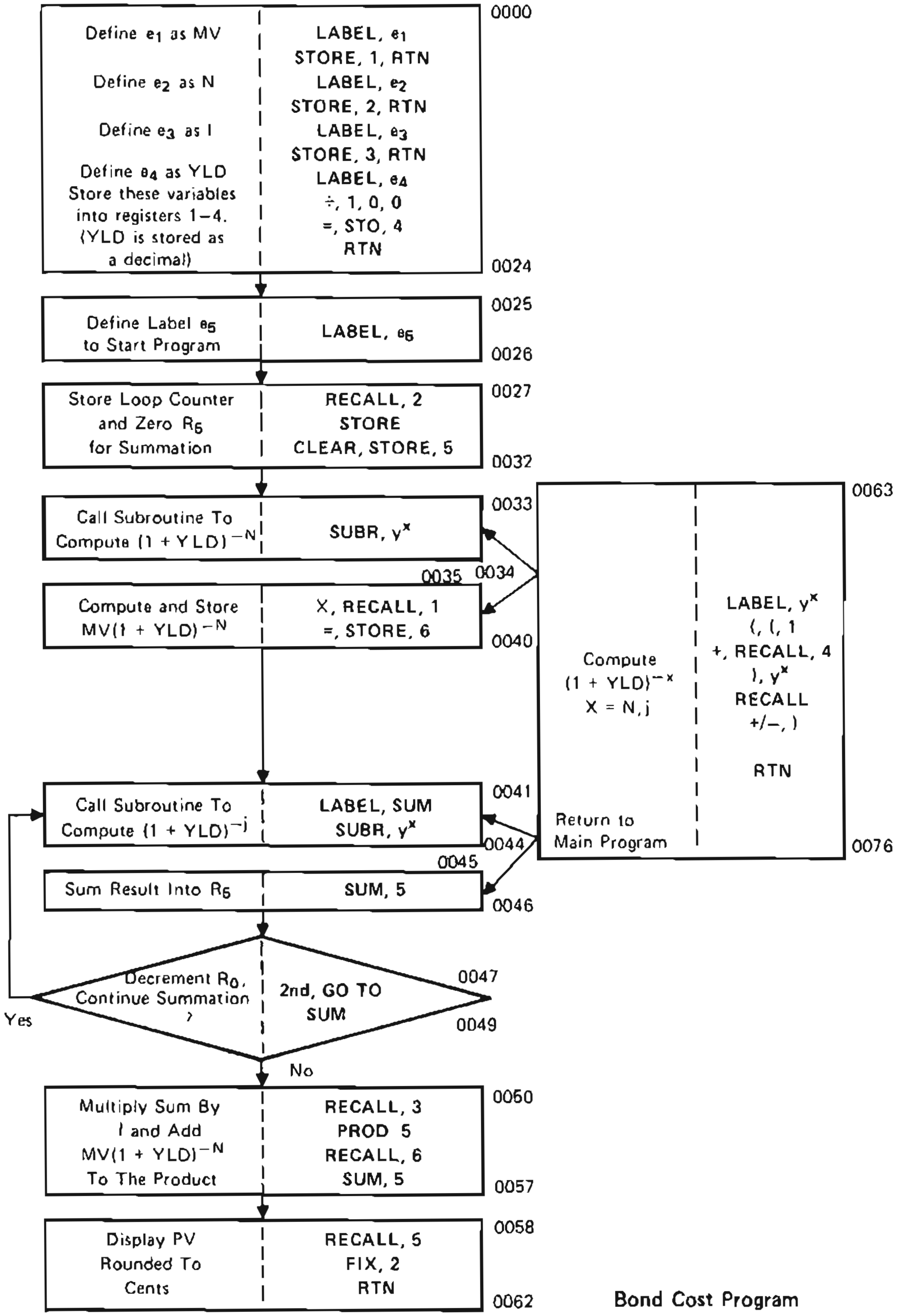
Many investors find buying bonds to be a secure and profitable means of putting their money to work. Others would be interested in buying bonds if they can analyze the potential earnings of their investments. Design a program that may be used to calculate the present value (cost) of a bond with periodic 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 CLEAR MEM	
2	Enter Learn Mode		Z LEARN	0000 Δ
3	Enter Bond Cost Program			
4	Exit Learn Mode		Z LEARN	0
5	Enter Maturity Value	MV	A e_1	MV
6	Enter Number of Periods	N	B e_2	N
7	Enter Coupon Value	I	C e_3	I
8	Enter Periodic Bond Yield to Maturity in Percent	YLD	D e_4	YLD/100
9	Compute Present Value		E e_5	PV
	Variables That Do Not Change Need Not Be Reentered For New Problems			

Location and Key Code		Key Sequence
0000	LBL	P LABEL
0001	e1	A e_1
0002	STO	STORE
0003	1	1
0004	RTN	$ RTN
0005	LBL	P LABEL
0006	e2	B e_2
0007	STO	STORE
0008	2	2
0009	RTN	$ RTN
0010	LBL	P LABEL
0011	e3	C e_3
0012	STO	STORE
0013	3	3
0014	RTN	$ RTN
0015	LBL	P LABEL
0016	e4	D e_4
0017	÷	÷
0018	1	1
0019	0	0
0020	0	0
0021	=	=
0022	STO	STORE
0023	4	4
0024	RTN	$ RTN
0025	LBL	P LABEL
0026	e5	E e_5
0027	RCL	RECALL
0028	2	2
0029	STO	STORE
0030	CLR	CLEAR
0031	STO	STORE
0032	5	5
0033	SBR	R SUBR
0034	y^x	y^x
0035	X	×
0036	RCL	RECALL
0037	1	1
0038	=	=
0039	STO	STORE
0040	6	6
0041	LBL	P LABEL
0042	SUM	SUM
0043	SBR	R SUBR
0044	y^x	y^x
0045	SUM	SUM
0046	5	5
0047	Π	2nd
0048	GTO	$ GO TO
0049	SUM	SUM
0050	RCL	RECALL
0051	3	3
0052	Π	PROD
0053	5	5
0054	RCL	RECALL
0055	6	6
0056	SUM	SUM
0057	5	5
0058	RCL	RECALL
0059	5	5
0060	FIX	FIX
0061	2	2
0062	RTN	$ RTN
0063	LBL	P LABEL
0064	y^x	y^x
0065	(	(
0066	(	(
0067	1	1
0068	+	+
0069	RCL	RECALL
0070	4	4
0071	)	)
0072	y^x	y^x
0073	RCL	RECALL
0074	+/–	+/–
0075	)	)
0076	RTN	$ RTN

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%.

Press	Display	Comments
20000 A	20000.	MV
12 B	12.	N
1400 C	1400.	I
8 D	0.08	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 X $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.

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

The roots x_1 and x_2 are found by: $x_1 = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$ $\qquad x_2 = \dfrac{-b - \sqrt{b^2 - 4ac}}{2a}$

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.

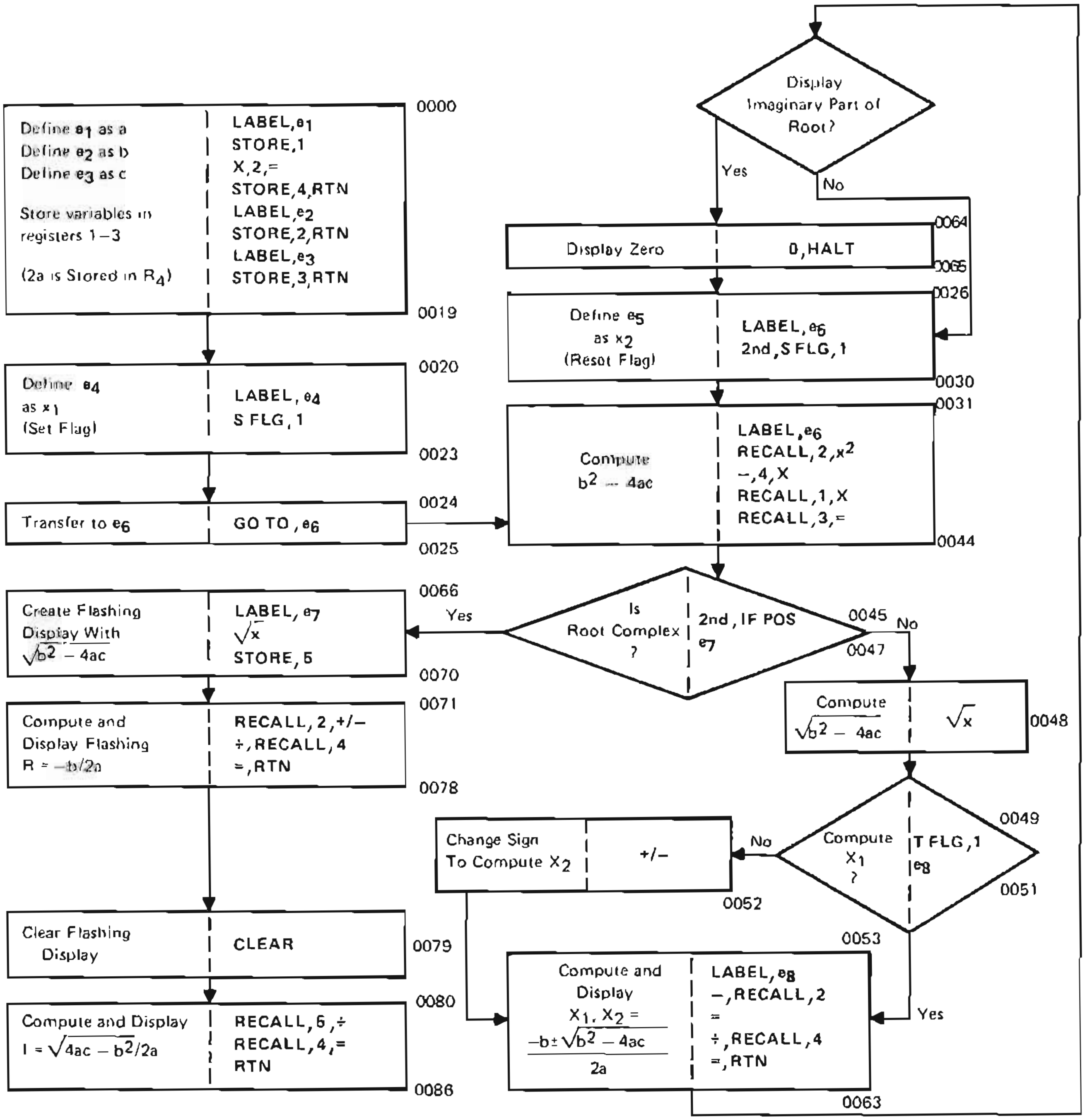
$x_1 = R + (i \cdot I)$ and $x_2 = R - (i \cdot I)$ where: $R = -b/2a$

$i = \sqrt{-1}$

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

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 RUN 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 equals has been used.



Quadratic Equation Program

USER INSTRUCTIONS

Step	Procedure	Enter	Press	Display
1	Clear Program Memory and Reset Program Pointer		2nd CLEAR MEM	
2	Enter Learn Mode		Z LEARN	0000 Δ
3	Enter Quadratic Equation Program			
4	Exit Learn Mode		Z LEARN	0
5	Enter a (a ≠ 0)	a	A e_1	a
6	Enter b	b	B e_2	b
7	Enter c	c	C e_3	c
8	Compute x_1 If Display Flashes Real Part – Root Is Complex		D e_4	x_1 (Real)
	– Compute Imaginary Part		SPACE RUN	x_1 (Imaginary)
9	Compute x_2 If Display Flashes Real Part – Root is Complex		E e_5	x_2 (Real)
	– Compute Imaginary Part		SPACE RUN	x_2 (Imaginary)

NOTE: If roots are real, there is no need to compute the imaginary part which is zero, if computed. If the roots are complex, the imaginary parts of x_1 and x_2 are equal. So, the roots are $x_1 = R + (i \cdot I)$.

Location and Key Code	Key Sequence	Location and Key Code	Key Sequence	Location and Key Code	Key Sequence
0000 LBL	P LABEL	0024 GTO	$ GO TO	0048 $\sqrt{x}$	$\sqrt{x}$
0001 e1	A e_1	0025 D/R	e_6 D/R	0049 TFS	J T FLG
0002 STO	STORE	0026 LBL	P LABEL	0050 1	1
0003 1	1	0027 e5	E e_5	0051 HYP	e_8 HYP
0004 X	×	0028 II	2nd	0052 +/–	+/–
0005 2	2	0029 SF	I S FLG	0053 LBL	P LABEL
0006 =	=	0030 1	1	0054 HYP	e_8 HYP
0007 STO	STORE	0031 LBL	P LABEL	0055 –	–
0008 4	4	0032 D/R	e_6 D/R	0056 RCL	RECALL
0009 RTN	S RTN	0033 RCL	RECALL	0057 2	2
0010 LBL	P LABEL	0034 2	2	0058 =	=
0011 e2	B e_2	0035 x^2	x^2	0059 ÷	÷
0012 STO	STORE	0036 –	–	0060 RCL	RECALL
0013 2	2	0037 4	4	0061 4	4
0014 RTN	S RTN	0038 X	×	0062 =	=
0015 LBL	P LABEL	0039 RCL	RECALL	0063 RTN	S RTN
0016 e3	C e_3	0040 1	1	0064 0	0
0017 STO	STORE	0041 X	×	0065 HLT	? HALT
0018 3	3	0042 RCL	RECALL	0066 LBL	P LABEL
0019 RTN	S RTN	0043 3	3	0067 ARC	e_7 ARC
0020 LBL	P LABEL	0044 =	=	0068 $\sqrt{x}$	$\sqrt{x}$
0021 e4	D e_4	0045 II	2nd	0069 STO	STORE
0022 SF	I S FLG	0046 IF +	N IF POS	0070 5	5
0023 1	1	0047 ARC	e_7 ARC	0071 RCL	RECALL

Quadratic Equation Program

Location and Key Code	Key Sequence	Location and Key Code	Key Sequence	Location and Key Code	Key Sequence
0072 2	2	0077 =	=	0082 ÷	÷
0073 +/–	+/–	0078 RTN	S RTN	0083 RCL	RECALL
0074 ÷	÷	0079 CLR	CLEAR	0084 4	4
0075 RCL	RECALL	0080 RCL	RECALL	0085 =	=
0076 4	4	0081 5	5	0086 RTN	S RTN

Example: Find the roots of the equation: $1.5x^2 + 3.7x + 2.25 = 0$

Press	Display	Comments
1.5 A e_1	3.	a → 2a
3.7 B e_2	3.7	b
2.25 C e_3	2.25	c
D e_4	– 1.088036702	Compute x_1 (Stable Display Indicates Root is Real)
E e_5	– 1.378629965	Compute x_2

Find the roots of the equation: $x^2 + 2x + 17 = 0.$

Press	Display	Comments
1 CLEAR MEM A e_1	2.	a → 2a
2 B e_2	2.	b
17 C e_3	17.	c
D e_4	"–1. ?"	Compute Root (Flashing Display Indicates Roots are Complex — R Is Displayed)
SPACE RUN	4.	Compute I

Programming Considerations III

ADDITIONAL TECHNIQUES

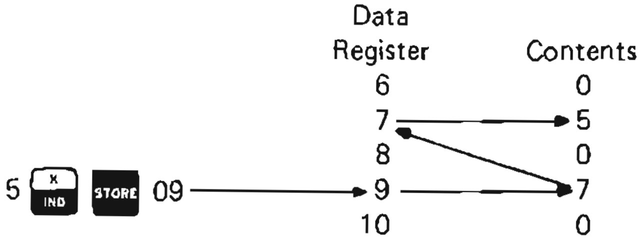
PROGRAMMING INDIRECT INSTRUCTIONS

A whole new set of capabilities can be added to data memory operations and transfer sequences through use of the indirect instruction, 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 are 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 directly by placing **IND** before 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 execution of the sequence below after 7 has been stored in R_9.

5 IND STORE 09



5 stored in register 7

This sequence goes to register 9 to find out where to store 5. It finds 7. So, 5 is stored in register 7.

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
0000 LBL	P LABEL	To enter X and press **e_1**
0001 e1	A e_1	
0002 STO	STORE	Store X in data register 0
0003 LBL	P LABEL	
0004 e2	B e_2	
0005 CLR	CLEAR	
0006 IND	K IND	
0007 STO	STORE	Zero is to be stored where register 0 says to
0008 II	2nd	
0009 GTO	R GO TO	DSZ loop in register 0
0010 e2	B e_2	
0011 RTN	S RTN	Go to e_2 if register 0 not zero Halts program when register 0 reaches zero

First time through the loop, X is in register 0 so the **CLEAR, IND, STORE** stores a 0 in register X. DSZ then decrements register 0 to (X − 1). Now the indirect store sequence stores its 0 in the 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?

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. Short-form addressing can and should be used for all transfer addressing except subroutines.

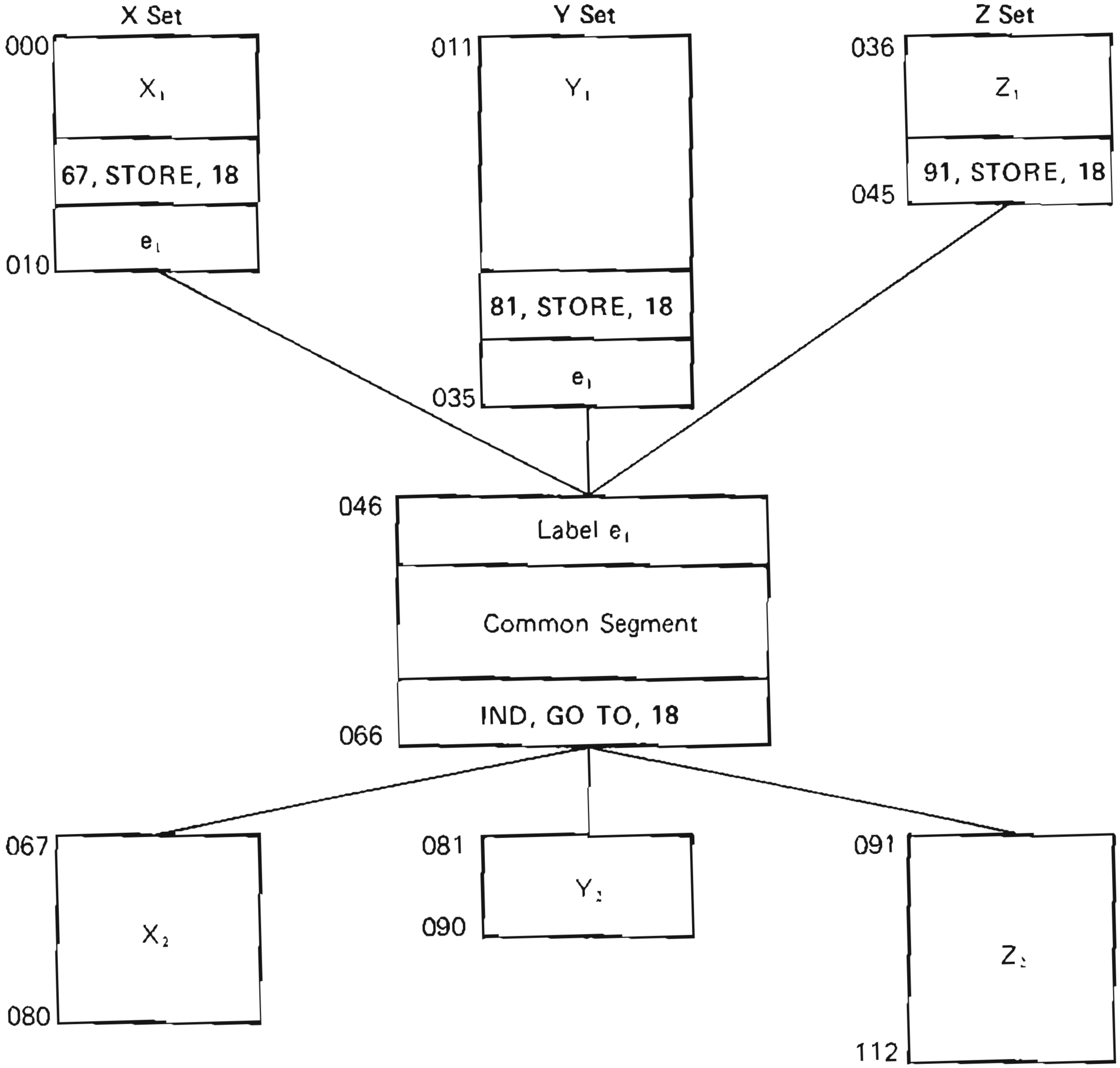
Indirect transfer sequences are begun by placing **IND** before either an unconditional transfer statement (**GO TO, SUBR**) or a conditional transfer instruction (**IF POS, 2nd IF ZRO,** etc.). The sequence must then be completed with the address of the data register containing the absolute address of the program location you wish to transfer to. Try this sequence from the keyboard.

Key Sequence	Display	Comments
35 STORE 18	35.	Store 35 in data register 18
IND GO TO 18 LEARN	0035 Δ	Program pointer sent to location 0035

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	Set Y	Set Z
X_1	Y_1	Z_1
e_1	e_1	e_1
X_2	Y_2	Z_2

The center portion (e_1) 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 e_1 at the ends of the segments X_1 and Y_1 to get e_1 (Z_1 flows directly to e_1), but how does the program appropriately transfer from e_1 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 e_1 and then end e_1 with a **IND GO TO** instruction. In the diagram, program locations are arbitrarily added to the beginning and end of each segment for illustrative purposes.


X Set
Y Set
Z Set
000
X_1
67, STORE, 18
e_1
010
011
Y_1
81, STORE, 18
e_1
035
036
Z_1
91, STORE, 18
045
046
Label e_1
Common Segment
IND, GO TO, 18
066
067
X_2
080
081
Y_2
090
091
Z_2
112

OTHER FEATURES

Indirect flag control is accomplished by placing the transfer address of the flag in a data register. For example, storing 6 in R_2 and completing the sequence **IND, T FLG, 1, 02** branches to location 0006 depending upon the status of flag 1. You cannot indirectly address the flag number itself, only the address.

	Data Register	Contents	
IND T FLG 1 02 →	02 →	0006	Transfer Address

For more on indirect addressing, see *Indirect Addressing* in Section IV.

III

Programming Considerations

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 **STORE** instruction is a good practice as the routine may be rerun without having to clear any data registers.

PROGRAMMING TECHNIQUES FOR MINIMIZING STEPS

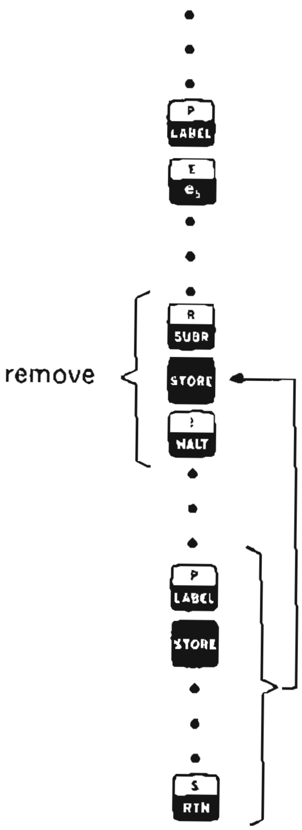
Condensing a program to a smaller number of steps is a time-consuming exercise. If a program fits within the program memory partition and operates properly, any time spent to condense the program, in most cases, is unnecessary except for the personal satisfaction of doing it.

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.

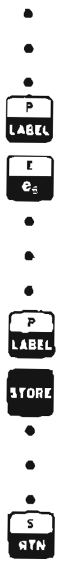
A program requiring numerous subroutines may still exceed the bounds of program memory. Optimization of subroutines thus becomes important.

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.

A program like this



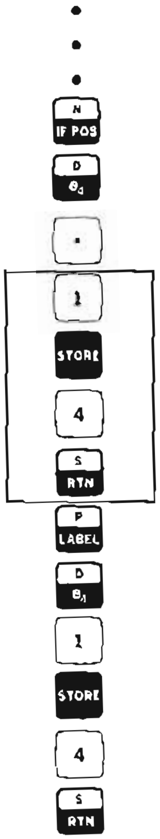
can look like this



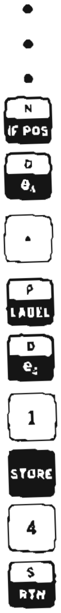
Not only is a savings of several steps realized, but one level of the subroutine return register has been freed. RTN now acts like a HALT, because the subroutine return register is clear.

As another illustration, consider the two sequences shown below.

Workable Segment



Efficient Segment



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 duplicated instructions enclosed in the box have been eliminated.

Programming Considerations

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.

Workable Segment

•
•
•
=
×
1
0
0
=
Int x
÷
1
0
0
=
•
•
•

Efficient Segment

•
•
•
LIMITED PRECISION
=
FIX
2
FIX
9
LIMITED PRECISION
•
•
•

The purpose and method of the routine on the left is fairly straightforward. The reasoning behind the second sequence is more efficient and more accurate because the display is rounded to the displayed digits.

III

Programming Considerations

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

```
•      •      •
•      •      •
•      •      •
+      +      +
1      1      4
0      EE     10ˣ
0      4      =
0      =      •
0      •      •
=      •      •
•      •
•
•
```

The relative efficiencies can be seen simply by counting program steps. 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 PROD, indirect instructions and the many single variable functions.

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.

Programming Considerations III

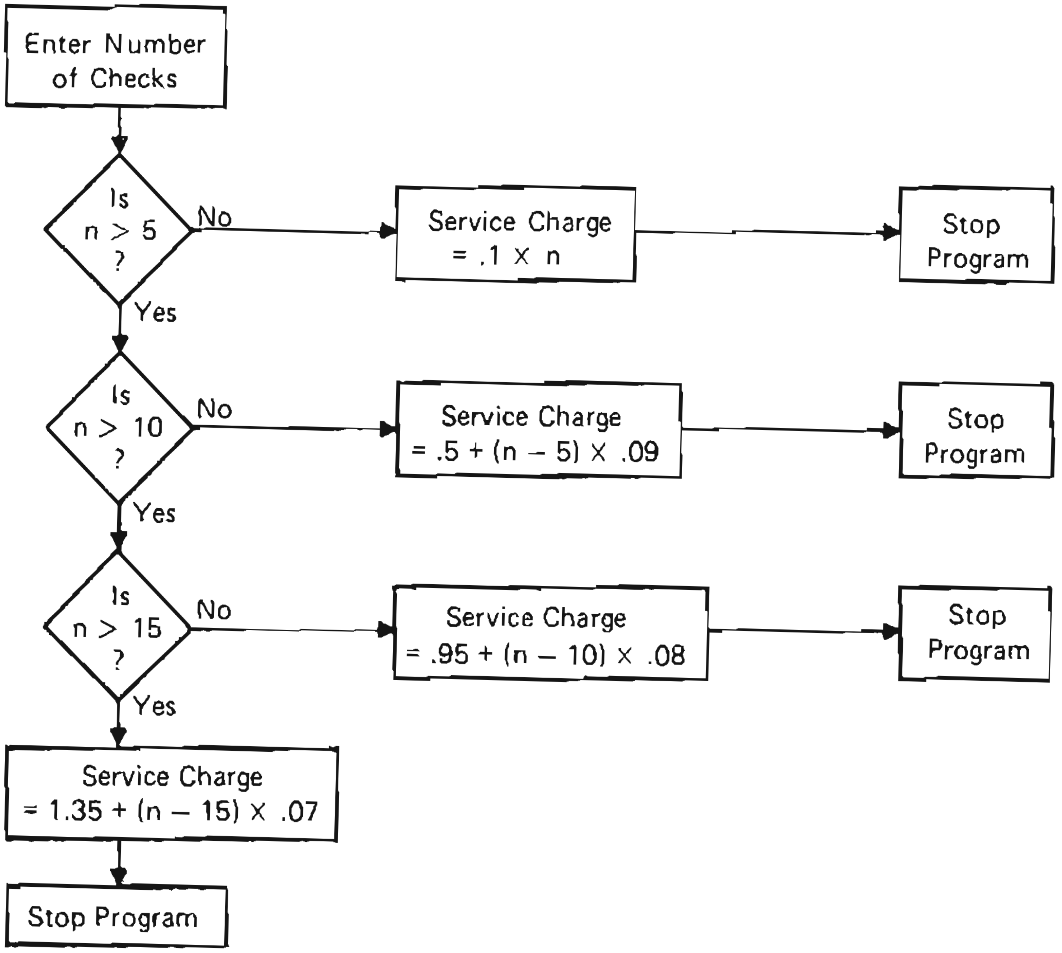
SERVICE CHARGE PROGRAM

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:

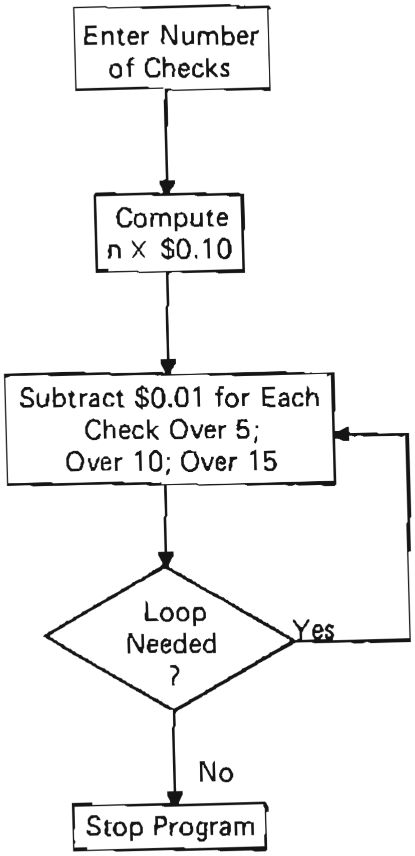
$0.10 per check for the first five checks (1-5),
$0.09 per check for the next five (6-10),
$0.08 per check for the next five (11-15),
$0.07 per check for each check over 15.

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



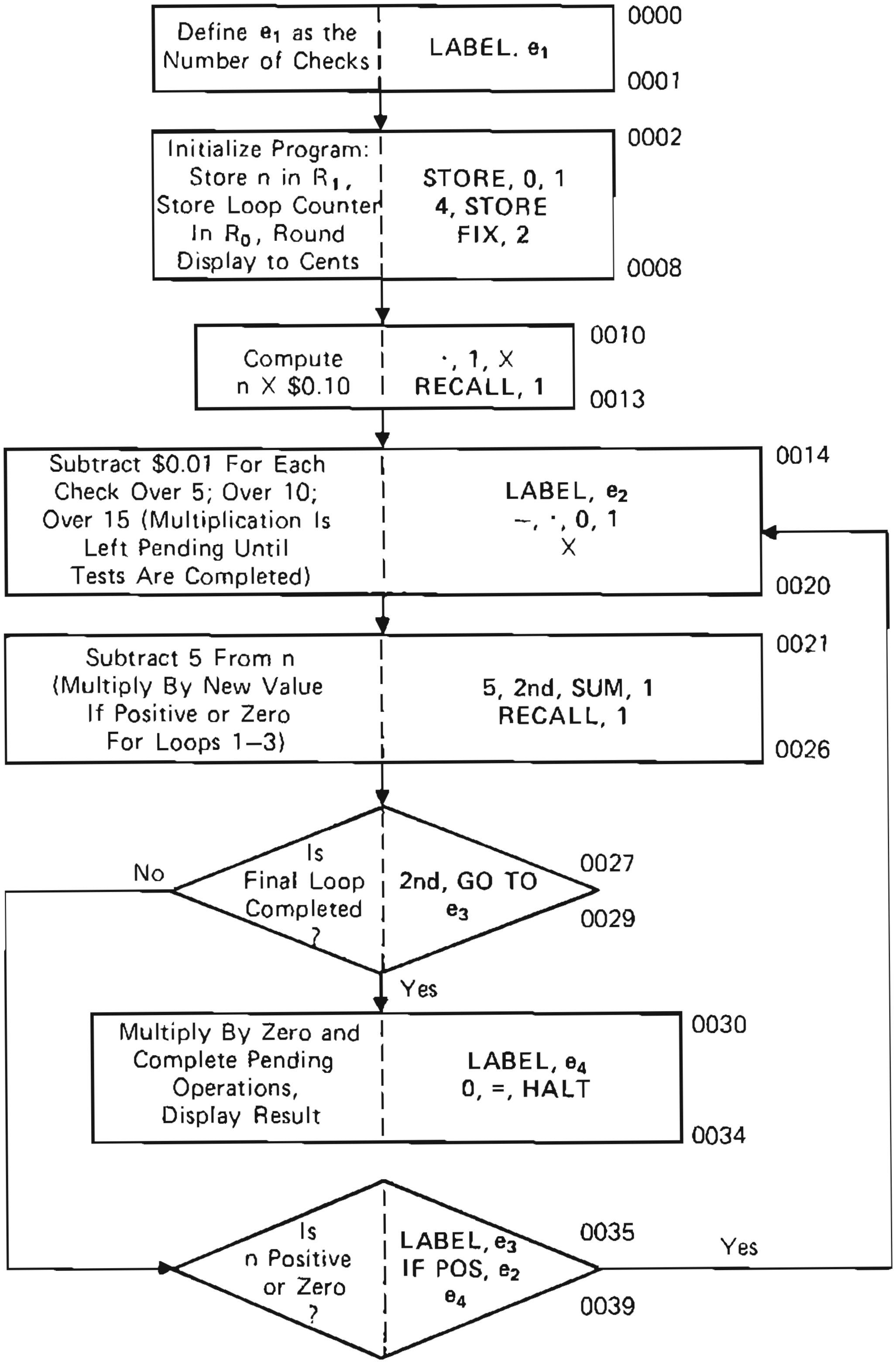
Service Charge Program (Basic Approach)

Attempting to write a program following this approach would probably require 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.


Define e_1 as the Number of Checks
LABEL, e_1
0000
0001
Initialize Program: Store n in R_1, Store Loop Counter In R_0, Round Display to Cents
STORE, 0, 1
4, STORE
FIX, 2
0002
0008
Compute n X $0.10
·, 1, X
RECALL, 1
0010
0013
Subtract $0.01 For Each Check Over 5; Over 10; Over 15 (Multiplication Is Left Pending Until Tests Are Completed)
LABEL, e_2
−, ·, 0, 1
X
0014
0020
Subtract 5 From n (Multiply By New Value If Positive or Zero For Loops 1–3)
5, 2nd, SUM, 1
RECALL, 1
0021
0026
Is Final Loop Completed ?
2nd, GO TO
e_3
0027
0029
No
Yes
Multiply By Zero and Complete Pending Operations, Display Result
LABEL, e_4
0, =, HALT
0030
0034
Is n Positive or Zero ?
LABEL, e_3
IF POS, e_2
e_4
0035
0039
Yes

Service Charge Program

The program is fairly straightforward until location 0021 where the multiplication in step 0020 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 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 ideal 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.

Programming Considerations III

Location and Key Code	Key Sequence	Location and Key Code	Key Sequence
0000 LBL	P LABEL	0020 X	×
0001 e1	A e_1	0021 5	5
0002 STO	STORE	0022 II	2nd
0003 0	0	0023 Σ	SUM
0004 1	1	0024 1	1
0005 4	4	0025 RCL	RECALL
0006 STO	STORE	0026 1	1
0007 FIX	FIX	0027 II	2nd
0008 2	2	0028 GTO	$ GO TO
0009 •	•	0029 e3	C e_3
0010 1	1	0030 LBL	P LABEL
0011 X	×	0031 e4	D e_4
0012 RCL	RECALL	0032 0	0
0013 1	1	0033 =	=
0014 LBL	P LABEL	0034 HLT	7 HALT
0015 e2	B e_2	0035 LBL	P LABEL
0016 –	−	0036 e3	C e_3
0017 •	•	0037 IF +	N IF POS
0018 0	0	0038 e2	B e_2
0019 1	1	0039 e4	D e_4

Service Charge Program

To run the program, simply key in some number of checks and press e_1. For instance, 1 check cost $0.10, 6 checks cost $0.59 and 63 checks cost $4.71. Because this program starts at location 0000, label e_1 could have been omitted and the program started by pressing QUE instead e_1.

ALPHANUMERICS IN A PROGRAM

Alphanumeric messages or labels can be built into programs to identify the variables and to label the calculated results. Section IV contains all the details on alphanumerics under *Program Prompting,* but a a message is placed in a program by pressing **ALPHA**, then keying in the letters you want and end with another **ALPHA**.

Now let's upgrade the Service Charge Program to label and print the number of checks. While you're at it, delete **LABEL, e_1**. Beginning at location 0005 insert the following:

```
PAPER ADV
PAPER ADV
ALPHA
N
U
M
B
E
R
SPACE
O
F
SPACE
C
H
E
C
K
S
ALPHA
PRINT
PRINT
```

Note that by inserting this message here, short-form addressing can be used for **STORE 1.**

Since the number of checks is to be printed out and this program works in FIX-2 format, place FIX 9 ahead of these alphanumerics.

Also, insert the following sequence beginning at old location 0034:

```
ALPHA
  S
  E
  R
  V
  I
  C
  E
SPACE
  C
  H
  A
  R
  G
  E
ALPHA
PRINT
PRINT
```

Listing the improved program shows the following.

Location and Key Code	Key Sequence	Location and Key Code	Key Sequence	Location and Key Code	Key Sequence
0000 STO	STORE	0027 STO	STORE	0054 =	=
0001 1	1	0028 FIX	FIX	0055 ALF	ALPHA ALPHA
0002 FIX	FIX	0029 2	2	0056 S	S RTN
0003 9	9	0030 •	•	0057 E	E e_5
0004 PA	PAPER ADV	0031 1	1	0058 R	R SUBR
0005 PA	PAPER ADV	0032 X	×	0059 V	V BSTEP
0006 ALF	ALPHA ALPHA	0033 RCL	RECALL	0060 I	I S FLG
0007 N	N IF POS	0034 1	1	0061 C	C e_3
0008 U	U STEP	0035 LBL	P LABEL	0062 E	E e_5
0009 M	M IF ERR	0036 e2	B e_2	0063	SPACE RUN
0010 B	B e_2	0037 –	–	0064 C	C e_3
0011 E	E e_5	0038 •	•	0065 H	H AUX
0012 R	R SUBR	0039 0	0	0066 A	A e_1
0013	SPACE RUN	0040 1	1	0067 R	R SUBR
0014 O	O IF ZRO	0041 X	×	0068 G	G WRITE
0015 F	F READ	0042 5	5	0069 E	E e_5
0016	SPACE RUN	0043 II	2nd	0070 ALF	ALPHA ALPHA
0017 C	C e_3	0044 Σ	SUM	0071 PRT	PRINT
0018 H	H AUX	0045 1	1	0072 PRT	PRINT
0019 E	E e_5	0046 RCL	RECALL	0073 HLT	J HALT
0020 C	C e_3	0047 1	1	0074 LBL	P LABEL
0021 K	K IND	0048 II	2nd	0075 e3	C e_3
0022 S	S RTN	0049 GTO	$ GO TO	0076 IF +	N IF POS
0023 ALF	ALPHA ALPHA	0050 e3	C e_3	0077 e2	B e_2
0024 PRT	PRINT	0051 LBL	P LABEL	0078 e4	D e_4
0025 PRT	PRINT	0052 e4	D e_4		
0026 4	4	0053 0	0		

Notice that the use of alphanumerics has considerably lengthened the program, but has made the use of it more meaningful and permanent by printing and labeling the values involved.

Now run the program using the same number of checks as before (1, 6 and 63), pressing QUE to start the program each time.

```
NUMBER OF CHECKS
       1.
 SERVICE CHARGE
      0.10

NUMBER OF CHECKS
       6.
 SERVICE CHARGE
      0.59

NUMBER OF CHECKS
      63.
 SERVICE CHARGE
      4.71
```

HOW TO MAKE PROGRAMS PROMPT YOU

In addition to providing messages and labels, the calculator can actually ask you questions or request you to enter certain things. Your responses back to the machine are made with one of the 5 gold keys at the top, center of the keyboard — **YES, NO, NOT APPLY, NOT KNOWN** or **ENTER**.

These instructions must, of course, be built into the program. The QUE key provides these special capabilities and is used as follows. First, key in alphanumeric message as before. Next, press QUE and enter four labels, one for each of the first four response keys. The ENTER response key simply skips all these labels and continues processing.

ALPHA Message ALPHA QUE	Label 1,	Label 2,	Label 3,	Label 4,	Normal instructions
	YES	NO	NOT APPLY	NOT KNOWN	ENTER

The calculator will display the message and stop and wait for your response. Responses act as shown above. If you press NO, transfer is immediately made to label 2 where processing resumes, etc.

III

Programming Considerations

To illustrate, further modify the service charge program to ask you for the number of checks. By rearranging things slightly and adding a few instructions, you can produce the following program.

Location and Key Code	Key Sequence
0000 4	4
0001 STO	STORE
0002 FIX	FIX
0003 9	9
0004 PA	PAPER ADV
0005 PA	PAPER ADV
0006 LBL	P LABEL
0007 Σ	SUM
0008 ALF	ALPHA ALPHA
0009 N	N IF POS
0010 U	U STEP
0011 M	M IF ERR
0012 B	B e
0013 E	E e
0014 R	R SUBR
0015	SPACE RUN
0016 O	O IF ZRO
0017 F	F READ
0018	SPACE RUN
0019 C	D e
0020 H	H AUX
0021 E	E e
0022 C	C e
0023 K	K IND
0024 S	S RTN
0025 ALF	ALPHA ALPHA
0026 QUE	Y QUE
0027 Σ	SUM
0028 Σ	SUM
0029 Σ	SUM
0030 Σ	SUM
0031 PRT	PRINT
0032 STO	STORE
0033 1	1
0034 FIX	FIX
0035 2	2
0036 •	•
0037 1	1
0038 X	×
0039 RCL	RECALL
0040 1	1
0041 LBL	P LABEL
0042 e2	B e
0043 −	−
0044 •	•
0045 0	0
0046 1	1
0047 X	×
0048 5	5
0049 II	2nd
0050 Σ	SUM
0051 1	1
0052 RCL	RECALL
0053 1	1
0054 II	2nd
0055 GTO	B GO TO
0056 e3	C e
0057 LBL	P LABEL
0058 e4	D e
0059 0	0
0060 =	=
0061 ALF	ALPHA ALPHA
0062 S	S RTN
0063 E	E e
0064 R	R SUBR
0065 V	V BSTEP
0066 I	I S FLG
0067 C	C e
0068 E	E e
0069	SPACE RUN
0070 C	C e
0071 H	H AUX
0072 A	A e
0073 R	R SUBR
0074 G	G WRITE
0075 E	E e
0076 ALF	ALPHA ALPHA
0077 PRT	PRINT
0078 PRT	PRINT
0079 HLT	2 HALT
0080 LBL	P LABEL
0081 e3	C e
0082 IF +	N IF POS
0083 e2	B e
0084 e4	D e

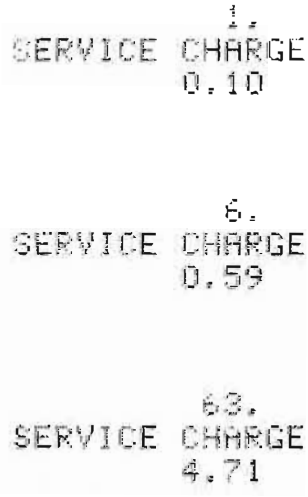
Paper advance at the start of a program takes slack out of the printer drive system which may result from previously tearing the paper off. If the drive system has slack, the characters in the first line printed may appear with flat tops or even as dashes.

In the program itself, instructions 0008–0030 ask you to enter the number of checks. When you key in the number of checks and press **ENTER**, the entered number is printed and the program continues. If you press any of the other response keys, the program follows the label assigned to that key after the QUE instruction and, in this case, loops back and asks for the number of checks again. This continues until the **ENTER** key is pressed.

The remainder of the program is the same, except for the 4 for the DSZ loop is stored at the first of the program instead of in the middle as before.

Now, instead of entering the number of checks at the beginning, just press QUE to start the program. The display soon asks for the number of checks to be entered. Now, you key in the number of checks and press **ENTER** and the program computes and labels the service charge. The number of checks is printed, but is not now labeled.

Running the program with the same number of checks (1, 6 and 63) as in the previous example, the following printout results.



```
                1.
SERVICE CHARGE
             0.10


                6.
SERVICE CHARGE
             0.59


               63.
SERVICE CHARGE
             4.71
```

III

Programming Considerations

STORING ALPHANUMERICS

Alphanumeric information can be saved and reused later by storing it and then recalling it from data registers. You can store 5 characters in each data register. By specifying a particular quarter of the display (as shown below), you can address the 5 characters in that quarter.

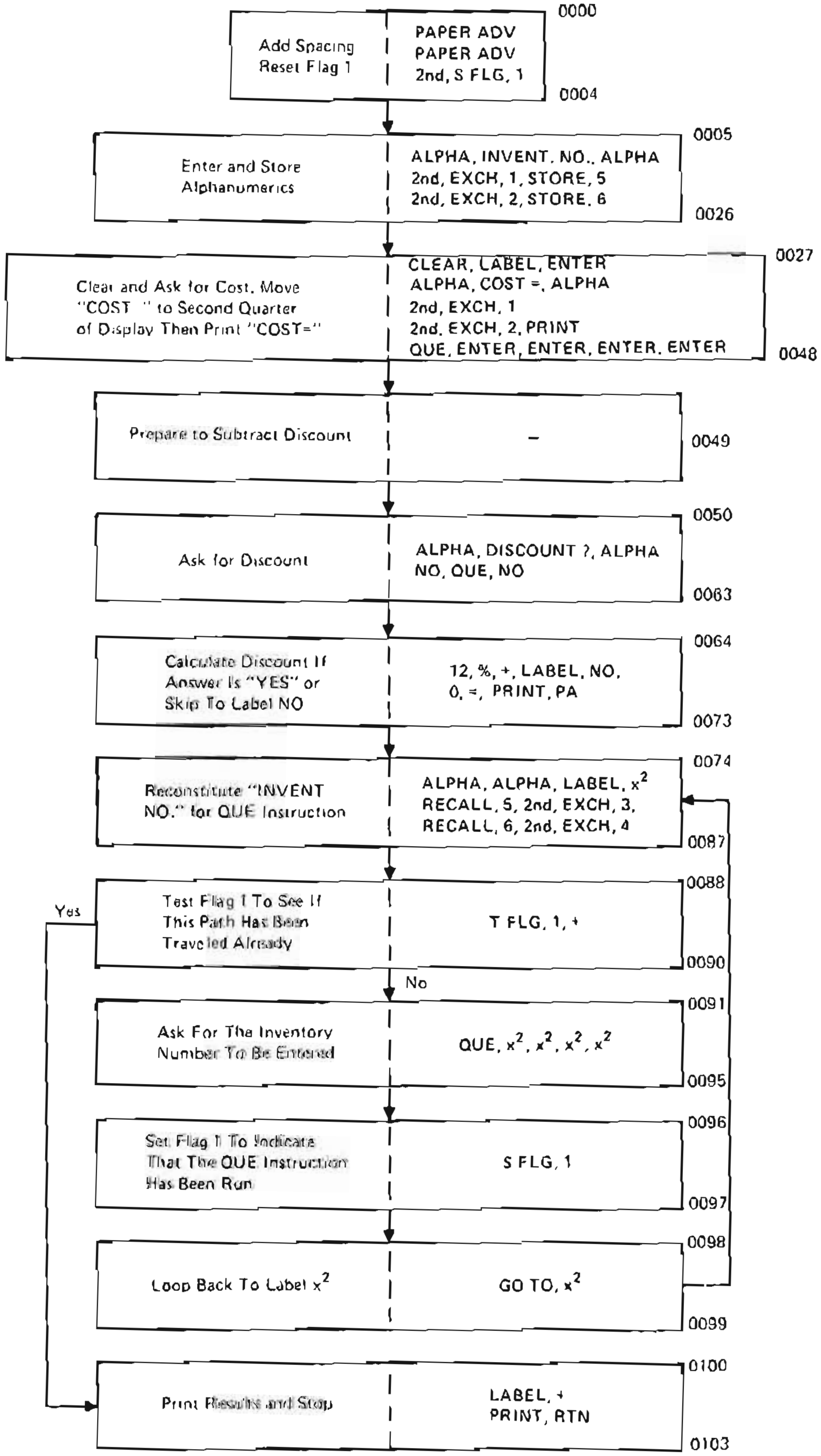
1 2 3 4 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	Character Positions
01	02	03	04	Sets of 5 characters

Key in the alphanumerics to be stored and press **2nd, EXCH 01** to retrieve the 5 left-most characters and convert them to numeric codes. Now they can be stored just like any other numbers. This procedure can be repeated for each quarter of the display – storing an entire 20-character message, 5 characters at a time, in 4 data registers. The number of digits required to specify the quarter is dependent on the partition.

To reconstitute the message in the display, simply reverse the process. First, recall the numeric codes for the alphanumerics for the first quarter. Now, press **2nd, EXCH 01** to convert the codes to alpha and place them in quarter 1. Recall the codes for each successive quarter and exchange them into place.

The numeric codes for the characters are discussed at length in the last part of Section IV.

Design a program to calculate the total cost of an item that may or may not have a 12% discount. The program prompts you for all information after pressing QUE to start. The inventory number of the item should be entered in alpha so that letters as well as numbers can be used.


0000
Add Spacing
Reset Flag 1
PAPER ADV
PAPER ADV
2nd, S FLG, 1
0004
0005
Enter and Store
Alphanumerics
ALPHA, INVENT. NO., ALPHA
2nd, EXCH, 1, STORE, 5
2nd, EXCH, 2, STORE, 6
0026
0027
Clear and Ask for Cost. Move
"COST " to Second Quarter
of Display Then Print "COST="
CLEAR, LABEL, ENTER
ALPHA, COST =, ALPHA
2nd, EXCH, 1
2nd, EXCH, 2, PRINT
QUE, ENTER, ENTER, ENTER, ENTER
0048
Prepare to Subtract Discount
–
0049
0050
Ask for Discount
ALPHA, DISCOUNT ?, ALPHA
NO, QUE, NO
0063
0064
Calculate Discount If
Answer Is "YES" or
Skip To Label NO
12, %, +, LABEL, NO,
0, =, PRINT, PA
0073
0074
Reconstitute "INVENT
NO." for QUE Instruction
ALPHA, ALPHA, LABEL, x²
RECALL, 5, 2nd, EXCH, 3,
RECALL, 6, 2nd, EXCH, 4
0087
0088
Yes
Test Flag 1 To See If
This Path Has Been
Traveled Already
T FLG, 1, +
0090
No
0091
Ask For The Inventory
Number To Be Entered
QUE, x², x², x², x²
0095
0096
Set Flag 1 To Indicate
That The QUE Instruction
Has Been Run
S FLG, 1
0097
0098
Loop Back To Label x²
GO TO, x²
0099
0100
Print Results and Stop
LABEL, +
PRINT, RTN
0103

III

Programming Considerations

Location and Key Code	Key Sequence
0000 PA	PAPER ADV
0001 PA	PAPER ADV
0002 II	2nd
0003 SF	I S FLG
0004 1	1
0005 ALF	ALPHA ALPHA
0006 I	I S FLG
0007 N	N IF POS
0008 V	V SSTEP
0009 E	I C_3
0010 N	N IF POS
0011 T	T QUE
0012 •	•
0013 N	N IF POS
0014 O	O IF ZRO
0015 •	•
0016 ALF	ALPHA ALPHA
0017 II	2nd
0018 XM	EXCH
0019 1	1
0020 STO	STORE
0021 5	5
0022 II	2nd
0023 XM	EXCH
0024 2	2
0025 STO	STORE
0026 6	6
0027 CLR	CLEAR
0028 LBL	P LABEL
0029 ENT	ENTER
0030 ALF	ALPHA ALPHA
0031 C	C 0_r
0032 O	O IF ZRO
0033 S	S RTN
0034 T	T QUE
0035 =	=
0036 ALF	ALPHA ALPHA
0037 II	2nd
0038 XM	EXCH
0039 1	1
0040 II	2nd
0041 XM	EXCH
0042 2	2
0043 PRT	PRINT
0044 QUE	T QUE
0045 ENT	ENTER
0046 ENT	ENTER
0047 ENT	ENTER
0048 ENT	ENTER
0049 –	–
0050 ALF	ALPHA ALPHA
0051 D	D C_d
0052 I	I S FLG
0053 S	S RTN
0054 C	C 0_r
0055 O	O IF ZRO
0056 U	U STEP
0057 N	N IF POS
0058 T	T QUE
0059 ?	? HALT
0060 ALF	ALPHA ALPHA
0061 NO	NO
0062 QUE	Y QUE
0063 NO	NO
0064 1	1
0065 2	2
0066 %	%
0067 +	+
0068 LBL	P LABEL
0069 NO	NO
0070 0	0
0071 =	=
0072 PRT	PRINT
0073 PA	PAPER ADV
0074 ALF	ALPHA ALPHA
0075 ALF	ALPHA ALPHA
0076 LBL	P LABEL
0077 x^2	x^2
0078 RCL	RECALL
0079 5	5
0080 II	2nd
0081 XM	EXCH
0082 3	3
0083 RCL	RECALL
0084 6	6
0085 II	2nd
0086 XM	EXCH
0087 4	4
0088 TFS	J T FLG
0089 1	1
0090 +	+
0091 QUE	T QUE
0092 x^2	x^2
0093 x^2	x^2
0094 x^2	x^2
0095 x^2	x^2
0096 SF	I S FLG
0097 1	1
0098 GTO	$ GO TO
0099 x^2	x^2
0100 LBL	P LABEL
0101 +	+
0102 PRT	PRINT
0103 RTN	S RTN

Notice that "INVENT. NO." is initially stored for later use as a prompting message and a label for printing. A loop beginning at location 0076 first retrieves "INVENT. NO." for use with the QUE prompting sequence. Upon repeating the sequence (from location 0099), the message is again reconstructed, but for printing purposes. The set flag signals the end of processing.

Once you have keyed the program into program memory, simply press **QUE** to start it. The cost is entered when asked for. The discount question needs a yes or no and the inventory number must be entered as an alpha message, 10 characters or less.

Calculate the cost of inventory item AJ-401C that initially costs $1000. Print the cost with and without the discount.

Press	Display	Print
QUE	COST=	COST=
1000 ENTER	DISCOUNT?	880.
YES	INVENT. NO.	
ALPHA AJ-401C ALPHA ENTER	0.	AJ-401C INVENT. NO.
QUE	COST=	COST=
1000 ENTER	DISCOUNT?	1000.
NO	INVENT. NO.	
ALPHA AJ-401C ALPHA ENTER	0.	AJ-401C INVENT. NO.

IV DETAILS

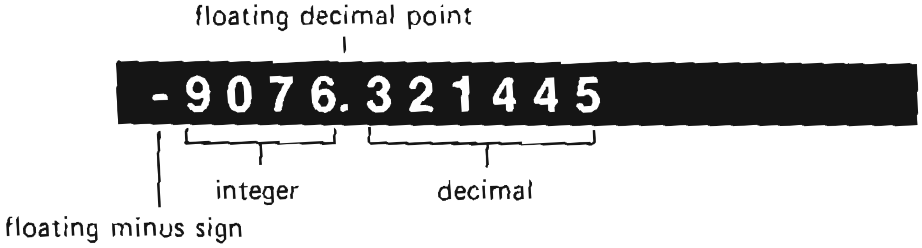
Now for an in-depth analysis of each facet of your calculator. This section is specifically designed as a detailed reference to be used once you have a basic understanding of the calculator's functions.

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

ENTERING, CLEARING AND DISPLAYING DATA

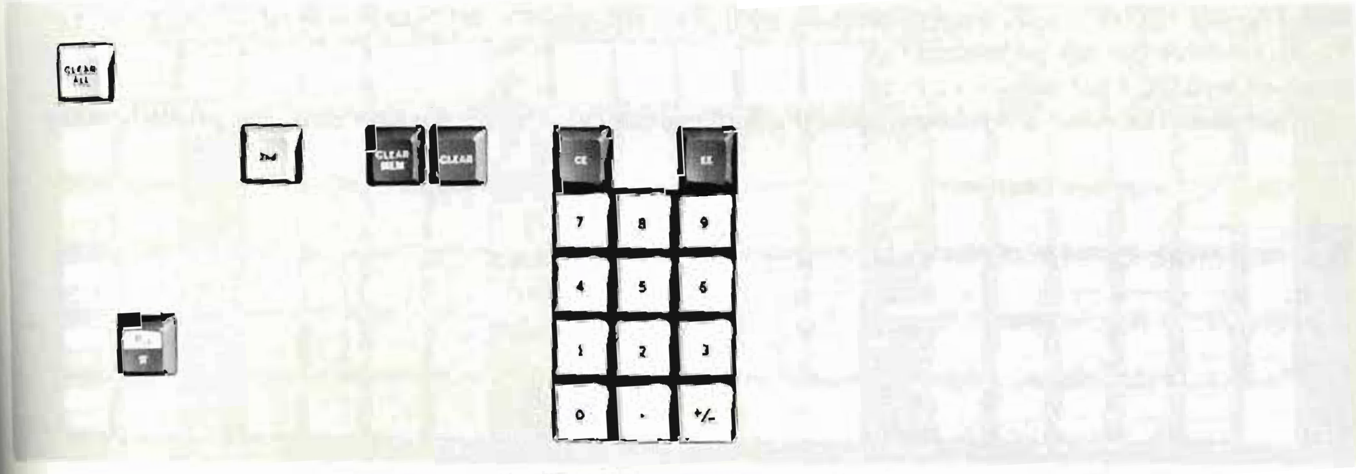
STANDARD DISPLAY

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. A numerical 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.

DATA ENTRY AND CLEARING KEYS



Number Entry & Clear Keys

Details

IV

The entry and clear operation keys, indicated above, are centrally located on the keyboard. The **2nd** key is also shown since it can be used as a prefix to the EE key and CLEAR MEM key. There are 24 key operations which can be modified by using the **2nd** key prefix. Each operation is described with the individual key descriptions in this manual. If the **2nd** key is pressed accidentally, it can be cancelled by pressing it a second time or by pressing a key that is not affected by it.

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.

[π] PI – Enters the value of pi (π) to 12 significant digits (3.14159265359) for calculations; display indicates the rounded value. **CE** does not remove π , 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.

[CE] CLEAR ENTRY – Clears entries made with the digit, decimal point and change sign keys only. To be effective it must be pressed before a function key. This key does not clear calculated results, numbers recalled from memory or π . CE also stops the flashing of the display when needed, but may not clear the error condition causing the flashing display. Use of this key does not affect pending operations.

[CLEAR] GENERAL CLEAR – Clears calculations in progress and the display. It resets scientific notation to standard format and stops a flashing display caused by an error condition. This key does not affect the contents of the data or program memories, angular mode, limited precision, fix-decimal display format or the partition.

The calculator effectively clears itself after most calculations. When the equals 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.

IV

Details

CLEAR ALL CLEAR ALL – Completely clears the calculator including all data registers and program memory. Results in display message "PROMPTING DESIRED?". This is a master clear key that clears everything.

CLEAR MEM CLEAR DATA MEMORY – Instructs the calculator to clear all data memory registers as defined by the current partition.

2nd CLEAR MEM CLEAR PROGRAM – Clears all locations of program memory, clears the subroutine return register, resets all flags and resets the program pointer to 0000 when pressed from the keyboard. **2nd, CLEAR MEM** acts as a reset when encountered in a program.

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: Enter 59263.04817

Press	Display
CLEAR ALL	PROMPTING DESIRED?
NO	0
59263.04817	59263.04817

Example: Enter –9126

Press	Display
CLEAR	0
9126 +/-	–9126

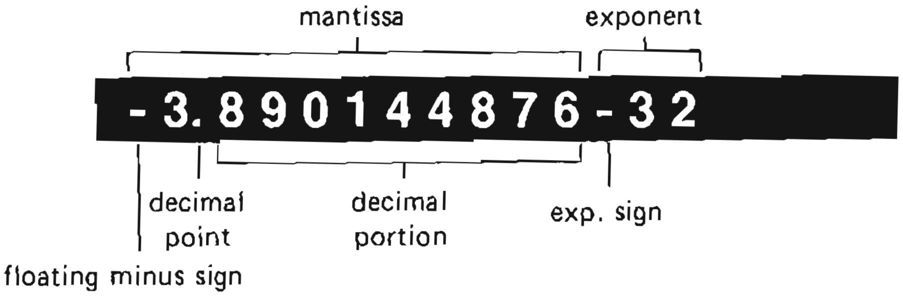
Example: 7.892 – π + (–2) = 2.750407346

Press	Display
7.892 −	7.892
θ14 π	3.141592654
+	4.750407346
2 +/-	–2
=	2.750407346

SCIENTIFIC NOTATION ENTRY

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 **CLEAR** or **CLEAR ALL** is pressed or until the calculator is turned off. Also, **2nd**, **EE** removes this format, but only if the displayed number is in the range ±5 X 10^{-11} to ±1 X 10^{10}. When **EE** is pressed *after a result* (intermediate or final), internal (guard) digits 11 and 12 are discarded and only the value in the display is used for the next calculation.

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 10 digits), press **EE**, then enter the exponent (any 2 digits).



This capability allows you to work with numbers as small as ±1 X 10^{-99} or as large as ±9.999999999 X 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 10 digits (and 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 X 10^{11} and can be entered into the calculator as:

Press	Display
CLEAR	0
3.2	3.2
EE	3.2 00
11	3.2 11

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 entry.

IV

Details

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 X 10^{20}

Press	Display
CLEAR	0
6025	6025
EE	6025 00
20	6025 20
+	6.025 23

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 X 10^{-12} then complete the decimal portion of the mantissa to read –4.96236 X 10^{12}.

Press	Display	Comments
CLEAR	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

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 CLEAR or **2nd**, EE 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
CLEAR	0
1.816 EE	1.816 00
3 −	1.816 03
581.432191 =	1.234567809 03
2nd EE	1234.567809

When **2nd**, 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 = 2nd EE	7.5 11
÷	7.5 11
25 = ÷	3. 10
25 =	1200000000.

If calculated results exceed 9999999999 or go below .0000000001, the display automatically goes into scientific notation. When this occurs without the EE having been pressed during the 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 X 1 EE = which multiplies the number in the display register by 1×10^0 and converts the display to scientific notation. The complete 12-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 the equals key in the middle of a computation. The reason is that the equals key completes all pending calculations. To avoid this, use these conversion methods only after computations are complete, or else multiply by X 1 EE, followed by another operation.

ARITHMETIC OPERATIONS

CALCULATOR HIERARCHY (AOS)

A universally accepted algebraic hierarchy governs the operational precedence of all mathematical operations. This hierarchy has been built into the SR-60A and is called the *Algebraic Operating System (AOS)* method of entry. As long as single operations are performed, with each operation completed by the equals key, the details of mathematical hierarchy within the calculator are relatively unimportant. For example, $2 + 3 = 5$, $4 \times 5 = 20$, $4 - 3 = 1$ and $9 \div 3 = 3$ as individual problems are entered into the calculator just as they are written. The relevance of mathematical hierarchy, however, is more obvious when considering several operations in a series. Consider the expression:

$$4 \times 5 + 9 \div 3 - 4^3 = ?$$

The matter of pending operations quickly becomes a problem. If the expression is keyed into the calculator, will the + operation cause the 4 X 5 entries to be completed or will the X and + operations remain pending until the ÷ operation, etc.? The solution to this situation is simply to establish a fixed processing precedence for each operation. The processing precedence of the SR-60A is simply the algebraic hierarchy.

The precedence established by algebraic hierarchy is summarized in six steps or processing levels. The steps are listed in ascending order of ability to complete pending operations.

Processing Level	Operation
1	Algebraic Functions and Conversions (except percent difference Δ%)
2	Percent Difference Δ%
3	Powers y^x and Roots $\sqrt[x]{y}$
4	Multiplication × and Division ÷
5	Addition + and Subtraction −
6	Equals =

Details

IV

The first-level operations act on displayed numbers only and will not complete pending operations. Since the first-level operations cannot have pending operations, the percent-difference operation has only the ability to complete another percent-difference operation. Third-level operations will complete pending power, root or percent-difference operations without affecting any pending fourth or fifth-level operations, etc. Of course, the equals operation is the most powerful because it will complete all pending operations. An important fact about the processing levels is that if operations from more than one level are pending when a higher level operation is entered, each pending operation is sequentially evaluated in the order in which the processing levels are listed.

With the precedence levels of algebraic hierarchy established, it is a simple matter to predict how the calculator will process the expression $4 \times 5 + 9 \div 3 - 4^3 =$. The evaluation process is shown by inserting phantom, or implied, parentheses.

$$(4 \times 5) + (9 \div 3) - (4^3) = -41$$

$$20 \quad + \quad 3 \quad - \; 64 = -41$$

To summarize how the implied parentheses were determined, first consider 4×5 is entered as written. What happens when + is entered? Since × is a fourth-level operation, the fifth-level + operation will evaluate 4×5 to be 20. After the 9 is entered, the pending operation is $20 + 9$. When the ÷ operation is entered, the + operation remains pending since a fourth-level operation cannot complete a fifth-level operation. Upon entering the 3, the pending operations are $20 + 9 \div 3$. When the − operation (fifth level) is entered, both pending operations are completed with respect to their processing level: $9 \div 3$ is first evaluated to be 3, followed by $20 + 3$ which equals 23. With the 23− operation now pending, entering the 4 and y^x operation leaves the 23− pending since y^x is a third-level operation. Upon entering the 3 and = , all pending operations are completed with respect to their processing level: 4^3 is first evaluated to be 64, followed by $23 - 64$ which completes the total expression with a result of −41.

PARENTHESES

The algebraic hierarchy precedence which controls the status of pending operations is very convenient for entering many types of mathematical problems into the calculator. There are however, other forms of problems which do not conform easily to those rules. For example,

$$4 \times (5 + 9) \div (7 - 4)^{(2+3)} = ?$$

If the SR-60A were equipped only with the mathematical operation precedence described thus far, the above expression would require careful step-by-step evaluation. In addition, the left-to-right entry format could not be followed. However, through the use of the parenthesis keys, the left-to-right algebraic entry format can be followed for the complete expression. The parentheses allow you to enter serial expressions without looking ahead to find the inner-most part of the expression. The calculator finds it for you, thus saving time in entering and reducing chances in making entry errors.

If there is ever doubt in remembering the precedence of the mathematical operations, actual parenthesis entries can be made right over the implied parenthesis previously described and the calculator will still evaluate the expression properly.

IV

Details

ARITHMETIC KEYS

The basic mathematical keys shown below are directly involved with the Algebraic Operating System entry method as described in the following definitions and examples.

(OPEN PARENTHESIS – Alters the normal processing precedence by holding all preceding operation entries pending until a corresponding close-parenthesis entry (or equals key) is used. Up to nine pending open parentheses are possible. An open-parenthesis entry is considered pending until a corresponding close parenthesis is entered or the equals key is used. More than nine pending open parentheses will cause a flashing display with a question mark.

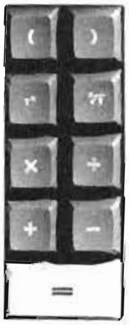
) CLOSE PARENTHESIS – Completes all pending mathematical operations entered since the last open-parenthesis entry and displays the result. If no intermediate results are desired, the equals key will supply any missing close parentheses and evaluate the total expression.

y^x $\sqrt[x]{y}$ POWER AND ROOT – Since the last open-parenthesis entry, either of these operations first completes any pending Δ% operation, then completes any pending y^x or $\sqrt[x]{y}$ operation, and finally, instructs the calculator to hold the entry as a pending operation. In either case the y value is the displayed number preceding the entry and the following entry is the x value. Note: $y \geqslant 0$.

× ÷ MULTIPLICATION AND DIVISION – Since the last open-parenthesis entry, either of these operations first completes any pending Δ% operation, then completes any pending y^x or $\sqrt[x]{y}$ operation, then completes any pending X or ÷ operation, and finally, instructs the calculator to hold the entry as a pending operation.

+ – ADDITION AND SUBTRACTION – Since the last open-parenthesis entry, either of these operations first completes any pending Δ% entry, then completes any pending y^x or $\sqrt[x]{y}$ operation, then completes any pending X or ÷ operation, then completes any pending + or – operation, and finally, instructs the calculator to hold the entry as a pending operation.

= EQUALS – Completes all pending operations in the same sequence specified for the + and – keys. Automatically supplies any missing close parentheses required to complete the calculation.



Arithmetic, Power, Root and Parentheses Keys

Example: 12.32 + 8.91 − 16.25 = 4.98

Press	Display
12.32 +	12.32
8.91 −	21.23
16.25 =	4.98

Example: 2.3 × 6.8 ÷ 6 = 2.606666667

Press	Display
2.3 ×	2.3
6.8 ÷	15.64
6 =	2.606666667

When each problem is ended with the equals key, there is no need to use the **CLEAR** key between problems.

Example: 8^3 = 512

Press	Display	Comments
8 y^x	8.	
3 =	512.	Note the use of = to complete the operation.

Example: $\sqrt[4.7]{512} = 3.770860074$

$\sqrt[4.7]{-512} = ?$

Press	Display	Comments
512 $\sqrt[x]{y}$	512.	
4.7 =	3.770860074	
512 +/- $\sqrt[x]{y}$	−512.	
4.7 =	3.770860074 ?	Flashing display

As a result of the minus sign under the radical, the display will flash. The flashing number displayed is the same as the result obtained without the minus sign.

Example: $8^3 \div 3 = 170.6666667$

Press	Display	Comments
8 y^x	8.	
3 ÷	512.	Completes pending power
3 =	170.6666667	

Example: $2^5 \times 6 \div 15 = 12.8$

Press	Display	Comments
2 y^x	2.	
5 ×	32.	Completes pending power
6 ÷	192.	Completes pending multiplication
15 =	12.8	

Example: (6 X 5) + (8 ÷ 2) = 34

Press	Display
6 ×	6.
5 +	30.
8 ÷	8.
2 =	34.

This example illustrates the *Algebraic Operating System* entry method. The same answer would result if the parentheses had been entered.

Press	Display
(	34.
6 ×	6.
5) + (	30.
8 ÷	8.
2)	4.
=	34.

The 34 in the display on the first line is the result of the previous solution. The 6 entry replaces the 34 on the next line. Also note that the result would be the same if the last close parenthesis were replaced with the equals key.

Now consider a more complex example which illustrates use of the *AOS* entry method and parentheses.

Example: $\frac{(2 \times 3) + (4 \times 5)}{(3 \times 4) + (5 \times 6)} = [(2 \times 3) + (4 \times 5)]/[(3 \times 4) + (5 \times 6)] = 0.619047619$

Press	Display	Comments
2 ×	2.	
3 +	6.	
4 ×	4.	
5 =	26.	Numerator
÷ (	26.	The range of (is the whole denominator
3 ×	3.	
4 +	12.	
5 ×	5.	
6 =	0.619047619	

The following example does not readily conform to the rules of algebraic hierarchy, thus, parentheses are essential in allowing left-to-right entry.

Example: $\frac{(2+3) \times (4+5)}{(3+4) \times (5+6)} = [(2+3) \times (4+5)]/[(3+4) \times (5+6)] = .5844155844$

Press	Display
(	Last result
2 +	2.
3) × (	5.
4 +	4.
5) ÷ ((	45.
3 +	3.
4) × (	7.
5 +	5.
6 =	.5844155844

Details

IV

Example: $\frac{2 \times 3}{4} + \frac{2^3 \times 4}{5} + \frac{\sqrt[4]{81} \times 5}{10} = 9.4$

Press	Display
2 ×	2.
3 ÷	6.
4 +	1.5
2 y^x	2.
3 ×	8.
4 ÷	32.
5 +	7.9
81 $\sqrt[x]{y}$	81.
4 ×	3.
5 ÷	15.
10 =	9.4

Example: $(1 + (5 - 3)^{-3}) \times (32 - 24) = 9$

Press	Display
(	Last result
1 + (	1.
5 −	5.
3) y^x	2.
3 +/-) × (	1.125
32 −	32.
24 =	9.

IV

Details

The next examples are designed to show some of the power this calculator has in dealing with complex expressions. You may never need to evaluate expressions this complex; however, you should follow the keystrokes carefully in order to get a feel for the use of parentheses. Notice that it does not hurt anything to use more parentheses than needed.

Example: $3^{(3/5)^{1/6}} \times (4 \times .93)^{.95} = 9.554263768$

Press	Display
3 $\boxed{y^x}$ $\boxed{(}$ $\boxed{(}$	3.
3 $\boxed{\div}$	3.
5 $\boxed{)}$ $\boxed{\sqrt[x]{y}}$	0.6
6 $\boxed{)}$ $\boxed{\times}$ $\boxed{(}$	2.742719751
4 $\boxed{\times}$	4.
.93 $\boxed{)}$ $\boxed{y^x}$	3.72
.95 $\boxed{=}$	9.554263768

The two parentheses on the first line were required to hold the y^x instruction pending while completing the $\sqrt[x]{y}$ instruction. Notice that even though the above examples are complex, none requires more than two levels of parentheses.

The next two examples require the calculator to hold ten levels of pending instructions before returning a result. These examples were constructed to fill the internal processing registers and should rarely occur as practical problems.

Example: $3(8 + 4(9 + 6(5 + 4(6 + 2(3 + 7))))) = 7980$

Press	Display
3 × (	3.
8 +	8.
4 × (	4.
9 +	9.
6 × (	6.
5 +	5.
4 × (	4.
6 +	6.
2 × (	2.
3 +	3.
7 =	7980.

Example: $10 + (9 \times 8^{(7-(6\div 5^{(4-(3\div 2^{(2+3)}))}))})$ = 18441278.44

Press	Display
10 +	10.
9 ×	9.
8 y^x (	8.
7 −	7.
6 ÷	6.
5 y^x (	5.
4 −	4.
3 ÷	3.
2 y^x (	2.
2 +	2.
3 =	18441278.44

The next example illustrates a problem with nine levels of pending parentheses which is also unlikely to occur as a practical problem.

Example: (((((((((π + 1) X (π + 2)) X (π + 3)) X (π + 4)) X (π + 5)) X (π + 6)) X (π + 7)) X (π + 8)) X (π + 9) = 95367733.95

Press	Display	Comments
((nine times)	Last result	
π +	3.141592654	Calculate 1st term
1) × (	4.141592654	
π +	3.141592654	Multiply 2nd term
2)) × (	21.29438236	
π +	3.141592654	Multiply 3rd term
3)) × (	130.7814223	
. .	.	Multiply 4th thru 8th terms
. .	.	
. .	.	
. .	.	
π +		Multiply 9th term for result
9 =	95367733.95	

HANDLING OPERATION ENTRY ERRORS

Function entries that require two operands (+, –, X, ÷, y^x, $\sqrt[x]{y}$, and Δ%) cannot be negated by pressing another function key. Pressing two such function keys in succession will cause a flashing display. Also, following any of these functions with = or) , or preceding with (, will produce the same result.

For short calculations, function entry errors are best handled by pressing **CLEAR** and starting over. When long calculations are involved, it is sometimes possible to complete the erroneous entry using 1 or 0 or another key sequence such that it does not affect the pending operations. The following chart indicates the possible actions that could be taken for some errors. To use the chart, locate the key pressed in error in the left column. Then follow that row across the chart to the column which is identified with the correct or intended key entry. The square at the point the row and column intersect shows a key or key sequence which may be used to precede the entry of the correct function without affecting pending operations. The points marked "CLEAR" indicate the **CLEAR** key should be used to start the problem over.

FUNCTION ENTRY ERROR	DESIRED FUNCTION ENTRY		
	+ OR –	X OR ÷	y^x OR $\sqrt[x]{y}$
+ OR –	0	CLEAR	CLEAR
X OR ÷	1	1	CLEAR
y^x OR $\sqrt[x]{y}$	1	1	1
(	CE,)	CE,)	CE,)

If one or more succeeding operands are the same as the first operand, the **CE** key may be used to reestablish the first operand as the second, third, etc. operands.

Example: $5 \times (5 + 5^5) = 15650$

Press	**Display**
5 [×] [(]	5.
[CE] [+]	5.
[CE] [y^x]	5.
[CE] [=]	15650.

CALCULATIONS WITH A CONSTANT

Repetitive calculations with a constant number are easily solved by placing the calculator in the constant mode.

[x⇌K] CONSTANT MODE – Instructs the calculator to save the number in the display register as a constant number and to save the function key pressed as the repeating operation. Each subsequent entry followed by pressing the equals key will repeat the calculation with the constant number and the newly entered number. The constant mode is in effect until an arithmetic key or the **CLEAR** key is pressed. It is a safe practice to press **CLEAR** before starting each series of constant calculations.

IV

Details

The operations that can be performed as constant calculations are as follows.

Key Sequence	Operation Performed
+ n x⇌K	Adds n to each subsequent entry.
− n x⇌K	Subtracts n from each subsequent entry.
× n x⇌K	Multiplies each subsequent entry by n.
÷ n x⇌K	Divides each subsequent entry by n.
y^x n x⇌K	Raises each subsequent entry to the n^{th} power (y^n).
$\sqrt[x]{y}$ n x⇌K	Takes the n^{th} root of each subsequent entry ($\sqrt[n]{y}$).
Δ% n x⇌K	Calculates the percent difference between n and each subsequent entry $\left(\frac{n - x}{x} \times 100\right)$.

Example: Add 512 to each of the following numbers: 2, −5, 10.2 and 6 × 10^6.

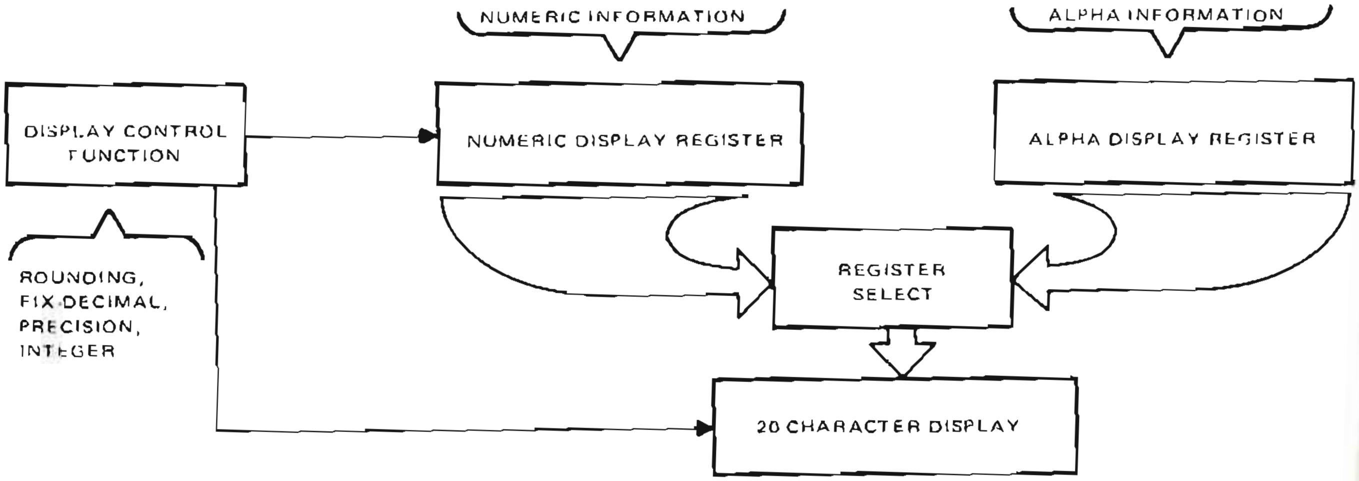
Press	Display
CLEAR	0
2 +	2.
512 x⇌K =	514.
5 +/- =	507.
10.2 =	522.2
6 EE	6 00
6 =	6.000512 06

DISPLAY INDICATIONS AND CONTROL

You should now be familiar with entering numbers into the calculator and performing the basic operations. To this point, displayed numbers have only been defined to be in standard display or scientific notation format as described at the beginning of this section. Perhaps you have noticed in the example problems that the tenth digit of a result can indicate that the result has been rounded. There are several display control functions. They allow you to control rounding, the number of decimal digits, the amount of precision, and you even have the capability to isolate a number into its integer or decimal fraction parts.

DISPLAY REGISTERS

The first consideration to be made about the display is the basic handling format the calculator uses to display information. The first step is the definition of terms. The display is the part of the calculator where you actually view the result. A *display register* is an internal element of the calculator which holds the information to be supplied to the display. As shown below, there are basically two different display registers which supply information to the display: a *numeric display register* and an *alphabetic (alpha) display register.*



Display Registers

The primary consideration at this point is the numeric display register since the alpha display register has no function in keyboard calculations. Therefore, reference to the "display register" in this manual normally implies the numeric display register. The latter part of this section describes some fundamental operations which involve the alpha display register.

IV

Details

The purpose in differentiating the display and the display register is that some of the display control functions affect only the display and some affect the display and the display register.

The nature of the display control functions is such that they are best described individually even though there can be considerable interaction between the functions. The keys which are related to display control are shown below and each function is described (with examples) separately for each reference.

FIX-DECIMAL CONTROL

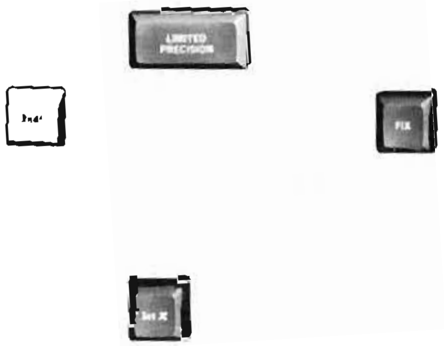
When the calculator is turned on or when the **CLEAR ALL** key is used, calculation results are displayed with up to ten significant digits with leading and trailing zeros suppressed. This means that the number of decimal places displayed (digits to the right of the decimal) will vary according to the individual calculation. The fix-decimal control allows the number of decimal places to be set at your option.

FIX n — **FIX-DECIMAL** — Instructs the calculator to display all results rounded to n number of decimal places where n is an integer from 0 through 8.

There are important facts to remember when exercising the fix-decimal control.

1. If the limited-precision control option has not been used, the display register will maintain results to full 12-digit capacity. Only the display is controlled by the **FIX** key. Subsequent calculations use the contents of the display register, not the display.

2. If the rounding control option has not been used, the fix-decimal result displayed is rounded up if the most significant nondisplayed digit is five or more and is rounded down when this digit is less than five.



Display Control Keys

3. The fix-decimal control option is functional in either standard display format or scientific-notation format.

4. If the **FIX** key is pressed and immediately followed by other than a numeral key, the calculator will assume **FIX 0**.

FIX 9 – **FIX-DECIMAL REMOVAL** – Returns the display to standard display format. The **CLEAR ALL** key will also remove the fix-decimal function. **CLEAR** or **CE** does not affect the fix-decimal mode.

Example: Sum the following amounts using **FIX 2** format.

$5.25, $25, $1.75, $.50

Press	Display
CLEAR FIX 2	0
5.25 +	5.25
25 +	30.25
1.75 +	32.00
.5 =	32.50

Example: Evaluate the following problem with the result rounded to three decimal places.

56 ÷ 95 X 2 = 1.179

Press	Display
CLEAR FIX 3	0
56 ÷	56.000
95 ×	0.589
2 =	1.179

(To display the complete result)

Press	Display
FIX 9	1.178947368

Using fix-decimal control with calculations in the scientific-notation format allows you to display a fixed number of the most significant nonzero digits without regard to the magnitude of the numbers. Since the mantissa is displayed with one integer digit, the fix-decimal selection is one less than the desired number of significant digits.

Example: Evaluate the following problem with the result showing only the three most significant digits.

$$\sqrt[5]{3.95 \times 10^{32}} = 3.31 \times 10^{6}$$

Press	Display
CLEAR FIX 2	0
3.95 EE	3.95 00
32 $\sqrt[x]{y}$	3.95 32
5 =	3.31 06

The last result can easily be converted to standard display format since the exponent is less than ten. The FIX 2 control can also be removed.

Press	Display
2nd EE	3306126.14
FIX 9	3306126.138

ROUNDING CONTROL

Unless otherwise directed, the calculator will automatically round the number in the display register if it contains more digits than can be displayed. This number is rounded up if the next, nondisplayable digit is 5 or more. There are two situations in which rounding will occur:

1. A calculation is internally processed to 12 digits and placed in the display register. This value is rounded to 10 digits for display.
2. A fix-decimal control limit is selected and the calculation is carried in the display register to more decimal places than selected for display.

Details

IV

2nd FIX n – ROUNDING SELECT – Selects the type of rounding desired where n is an integer which sets the weighting of the rounding process as follows:

n = 0: Rounds results down (i.e., discards the decimal portion)

n = 1: Add one to last displayable digit (round up) if next, nondisplayed digit is 9

n = 2: Add one to last displayable digit (round up) if next, nondisplayed digit is 8 or 9

n = 3: Add one to last displayable digit (round up) if next, nondisplayed digit is 7 or more

n = 4: Add one to last displayable digit (round up) if next, nondisplayed digit is 6 or more

n = 5: Add one to last displayable digit (round up) if next, nondisplayed digit is 5 or more (normal round off)

n = 6: Add one to last displayable digit (round up) if next, nondisplayed digit is 4 or more

n = 7: Add one to last displayable digit (round up) if next, nondisplayed digit is 3 or more

n = 8: Add one to last displayable digit (round up) if next, nondisplayed digit is 2 or more

n = 9: Add one to last displayable digit (round up) if next, nondisplayed digit is 1 or more

The rounding process does not affect the contents of the display register, only the display is affected. In addition, the rounding process is based only on the first nondisplayed digit in the display register to the right of the displayed number. The nondisplayed digits themselves are not rounded.

If the sequence **2nd, FIX** is immediately followed by other than a numeral key, the calculator will assume n = 0 (round down).

Press	Display	Comments
CLEAR FIX 0	0	Select fix 0 format (display no fractions)
2nd FIX 9	0	Select n = 9 rounding
1 +	1.	
.1 =	2.	Round 1.1 up to 2
2nd FIX 5	2.	Select n = 5 rounding
1.4 +	1.	
.1 =	2.	Round 1.5 up to 2
2nd FIX 0	2.	Select n = 0 rounding
1.8 +	1.	
.1 =	1.	Round 1.9 down to 1

IV

Details

LIMITED PRECISION CONTROL

The display control functions previously described have affected the display, but not the display register. For those functions, the full 12-digit capacity was used at all times. The limited precision control allows you to limit the contents of the display register to the same number that is in the display. The precision or accuracy of a calculation is limited to as few digits as desired by using the other display control functions with limited precision.

LIMITED PRECISION **LIMITED PRECISION** – Selects or cancels limited precision. A red light above the key is illuminated when limited precision is in effect. Limited precision operates only on *results* (produced by the equals key,) key, or any function or conversion key).

Example: Effects of limited precision with normal rounding.

Press	Display
CLEAR 2nd FIX 5	0
LIMITED PRECISION FIX 1	0
4.1654392 ×	4.2
1 =	4.2
FIX 7	4.2000000

Fix-decimal 7 was used to show that the nondisplayed digits were discarded when limited precision was selected.

Example: Evaluate the following problem such that the intermediate result is rounded to the nearest penny before final evaluation.

$95 ÷ 300 lb = $.32 per lb. X 50 lb = $16.00

Press	Display	Comments
LIMITED PRECISION FIX 2	0	Verify limited precision indication is on.
95 ÷	95.00	
300 =	0.32	Nondisplayed digits discarded.
×	0.32	
50 =	16.00	

The EE key has some limited-precision power on an individual result basis. The nondisplayed digits in the display register may be discarded at any time by pressing the EE key before pressing the next function key. Entering an exponent is optional. If the result is not desired in scientific-notation format, then follow the EE entry with **2nd**, **EE**.

Example: Solve the last problem again, except use the EE key to effect limited precision.

Press	Display	Comments
CLEAR LIMITED PRECISION	0	Verify limited precision indicator is off.
95 ÷	95.00	
300 = EE 2nd EE	.32	Nondisplayed digits discarded.
×	0.32	
50 =	16.00	

Of course this problem is easier using the **LIMITED PRECISION** key, but it is important to remember this possible use of the EE key.

INTEGER AND DECIMAL FRACTION CONTROL

It is occasionally desirable to conveniently discard all digits to the right or left of the decimal. The **Int x** key will quickly perform this operation.

Int x INTEGER — Instructs the calculator to discard the decimal-fraction digits of the number in the display and in the display register. The integer operation is performed directly on the display register and retains the true integer value without regard to display format.

2nd Int x INTEGER REMOVAL — Instructs the calculator to discard the integer digits with the conditions specified for the integer key.

Example: Determine the integer portion of 31 ÷ 9.

Press	Display
CLEAR FIX 9	0
31 ÷	31.
9 =	3.444444444
Int x	3.

Example: Determine the decimal-fraction portion of 31 ÷ 9.

Press	Display
31 ÷	31.
9 =	3.444444444
2nd Int x	.4444444444

Example: Determine the integer portion of 3.59463 X 10^3.

Press	Display
3.59463 EE	3.59463 00
3	3.59463 03
Int x	3.594 03

Example: Determine the decimal-fraction portion of 3.59463 X 10^3.

Press	Display
3.59463 EE	3.59463 00
3	3.59463 03
2nd Int x	6.3–01

NOTE: If you want the results of calculations rounded to the nearest integer for display, press **2nd FIX 0** before performing the calculations.

Details

IV

DISPLAYED RESULTS VERSUS ACCURACY

The basic mathematical limit of the SR-60A is the number of digits it uses for calculations. The calculator appears to use 10 digits as shown by the display, but actually uses 12 digits to perform calculations. The reason for this is obvious if you consider the following simple problem.

$$\frac{1}{3} \times 3 = 1$$

$$.3333333333 \times 3 = .9999999999$$

While you know the answer should be 1, the calculation result with decimal numbers turns out to be a decimal fraction when limited strictly to 10 digits.

If 11 digits were used to produce the 10-digit result, and the capability to round off is added, it is easy to see the decimal-fraction result would become a 1 as expected. The SR-60A has this round off capability and instead of 11 digits, it uses 12 digits to assure that the 10 displayed digits are as accurate as possible. The extra digits serve as guard digits to protect the accuracy of the 10 displayed digits. The higher order mathematical functions use iterative calculations. Therefore, the cumulative rounding error from the 12th digit is two orders of magnitude from the displayed 10th digit. In this way, the SR-60A presents normal results rounded accurately to 10 places.

The previously described display control functions allow you to control the rounding process and the existence of guard digits. These functions will work for you as indicated as long as you realize what they do and how they affect the displayed number. Unexpected results can occur, however, if you unknowingly operate the calculator without considering the display control functions. For example, if the person using the calculator before you was using round down (**2nd, FIX 0**), and you keyed in 1 ÷ 3 X 3 =, the answer displayed would be .9999999999 instead of 1. There are more extreme discrepancies possible if you are unaware of the control functions selected.

Example: 4 ÷ 3 X 3 ≠ 4

Press **FIX 0** and **2nd, FIX 0** before starting.

Press	Display	Comments
CLEAR	0	
4 ÷	4.	
3 ×	1.	
3 =	3.	Result is 3
FIX 9	3.999999999	Full result without rounding
2nd FIX 5 =	4.	Normal result

Another case where the correct result cannot be recovered is when limited precision is added to the two unknown factors above.

Example: (2.5 + 0.4) X 1000 ≠ 2900

Press **FIX 0, 2nd, FIX 0,** and **LIMITED PRECISION** before starting.

Press	Display
CLEAR (	0
2.5 +	2.
.4) ×	2.
1000 =	2000.
FIX 9	2000.
2nd FIX 5 =	2000.

Notice in this case the erroneous result cannot be corrected because limited precision discarded all nondisplayed digits when the right parenthesis and equals were pressed.

Either of two precautionary measures should be taken if you begin using the calculator when the power is already on.

1. Press **FIX 9, 2nd FIX 5** and check Limited Precision (indicator off)

2. Press **CLEAR ALL**

The **CLEAR ALL** key is the simplest method, however, it clears all data registers and the program memory and is not suggested as a standard practice.

The guard digits serve a very useful purpose in that, without them, the calculator would frequently show answers with an accuracy much less than ten digits. Normally, there is no need to consider the guard digits. On certain calculations, however, the guard digits may appear as the result when not expected. Without going into elaborate details, the internal processing of the mathematical functions are controlled to provide accurate results rounded to ten digits. Mathematical limits 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° ≠ 0

Press	Display	Comments
DEG MODE CLEAR	0	
45 SIN −	.7071067812	Sin 45°
45 COS	.7071067812	Cos 45°
=	1.–12	

Notice the nonzero result of the example is 1 X 10^{-12}. Compared to the operands of the calculation (.7071067812), the 1 in the result is in the 12th digit or last guard digit. The significance is that results which are smaller than any entry or intermediate result by a factor of 10^{-11} to 10^{-12} are potentially equal to zero. A case where this type of result would not be zero is the reciprocal of 1 X 10^{12} which is 1 X 10^{-12}.

Another example when an unexpected nonzero result may occur is when limited precision is activated on only part of the entries or intermediate results. A typical case involves use of the **EE** key.

Example: $\pi - \pi \neq 0$

Press	Display
CLEAR	0
π −	3.141592654
π EE	3.141592654 00
=	–4.1–10

As previously indicated, when the π key is used, the calculator internally uses 3.14159265359 though it only displays 3.141592654. Pressing π **EE**, however, causes the calculator to use only the displayed digits – discarding all digits not displayed. The calculation which took place was actually 3.14159265359 minus 3.141592654 which equals –0.00000000041 or –4.1 X 10^{-10}. The limited precision effect of the EE key only affected the second entry of π.

In any event, the operations described are not calculator design errors. The calculator must operate as described to allow the versatile interchange of numbers in standard display format and scientific-notation format.

IV

Details

PRINTER OPERATIONS

The printer is built into the SR-60A to provide you with a permanent record of your calculations. When running prerecorded programs, the printout tape is the primary source of results while the display normally presents prompting messages and instructions.

The paper used by the printer is a heat-sensitive type paper (thermal paper). The only mechanical part of the printer is a precision motor which moves the paper past the stationary electronic printheads. Small semiconductor elements are heated very quickly by electronic circuits to produce the numbers, letters and symbols you can read on the thermal paper.

Since the printer is basically an electronic device, using metal probes or other sharp objects may damage the printheads. Please refer to Appendix B, Maintenance and Service, for paper replacement instructions and for instructions about proper care of the printer. The diagram below shows the keys that control the printer.

Printer Control Keys

BASIC PRINTING

PAPER ADV PAPER ADVANCE – This key advances the printer paper without printing. If the key is pressed quickly, a single unwritten line is advanced. If the key is held down, the paper will continue to advance until the key is released. The paper advance instruction is a programmable instruction.

PRINT PRINT — This key causes the current contents of the display to be printed. If the content of the display is an alphanumeric message, it will be cleared from the display after printing.

TRACE TRACE — This key causes the calculator to enter the trace mode. In this mode every new function or result entered is automatically printed. Number entry keys do not cause a line to be printed. A number entry followed by a function will cause a line to be printed. When the **TRACE** key is pressed, an indicator comes on above the key and remains on until the key is pressed again.

When in the trace mode of operation, the printer provides a detailed record of numbers, function entries and results. Since the calculator must devote some amount of time to the printing process, it will ignore keyboard entries during the short printing periods following print instructions. Be careful not to make entries while the printer is operating because they will not be accepted until the printing is finished. The trace operation may also be controlled by flag 9 in a program. See *Flag Operation* later in this section.

Example: Use the trace mode to print out the following calculation.

2.65 + 3.95 = 6.6

Press	Display	Printout
CLEAR ALL NO	0	
TRACE	0	0
2.65 +	2.65	2.65 +
3.95 =		3.95 =
	6.6	6.6
CLEAR	0	6.6 CLR
TRACE	0 TRC	0 TRC

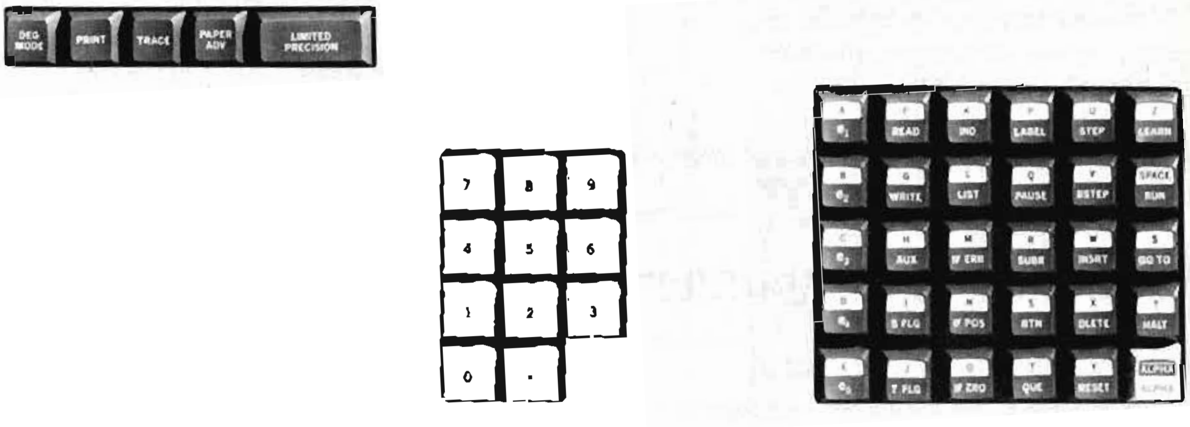
PRINTING ALPHANUMERIC MESSAGES

The alphanumeric capabilities of the SR-60A can be used to print out messages and labels along with the normal numbers and results of calculations.

The block of keys on the right side of the calculator (see opposing figure) are primarily programming keys. However, notice that the top of most have alphabetical letters or symbols.

IV

Details



Alpha and Numeric Entry Keys

The ALPHA key in the lower right-hand corner is the control key for activating the alphabetical functions of the other keys. The operation of the ALPHA key is very simple and is similar to the "SHIFT" key on a typewriter. Press it once, the alpha mode is established. Now, each key becomes an alphabetic or numeric symbol only, and all mathematical and programming functions of the keys are locked out. Pressing the ALPHA key again takes the calculator out of alpha mode and returns keys to their normal functions. Entering and exiting the alpha mode does not affect the number in the numeric display register or any pending operations. This is possible in the alpha mode because all entries are routed to the alpha display register without affecting the numeric display register. The CLEAR key may be used to exit the alpha mode and simultaneously clear the numeric display and processing registers.

For a simple example of the alpha mode, press the ALPHA key one time and press the A, B, C, D and E keys. Notice that the A key caused an A to appear in the left-most digit of the display, the B key placed a B in the second digit, etc. Continue by pressing keys, F, G, H, I, J, K, L, M, N, O, P, Q, R, S and T. The display shows 20 letters which is the maximum number of characters which may be entered at a time. Press the ALPHA key again and the calculator is out of the alpha mode even though the contents of the alpha display register are still displayed. Pressing the PRINT key will print the letters in the display and the display automatically reinstates the contents of the numeric display register.

Note that all other keys (except CLEAR and CLEAR ALL) produce letters or a symbol representative of the key function when in the alpha mode. The DEG MODE, PRINT, TRACE, PAPER ADV and LIMITED PRECISION keys are assigned special symbols and punctuation marks in the alpha mode. DEG MODE produces a degree symbol, PRINT produces a slash mark, TRACE produces an asterisk, PAPER ADV produces a comma, and LIMITED PRECISION produces an apostrophe. The key marked SPACE produces one blank digit each time it is pressed.

If an error is made while in the alpha mode, press **ALPHA** to exit the alpha mode and start over. When operating the printer in the trace mode, the alpha message will be automatically printed when the **ALPHA** key is pressed to exit the alpha mode. If you want to return the display to the numeric display register without printing or without affecting the contents of the numeric register, press a nonmathematical function key such as the **YES** or **CE** key.

A detailed use of alphanumerics is included in *Program Prompting Functions* later in the programming section.

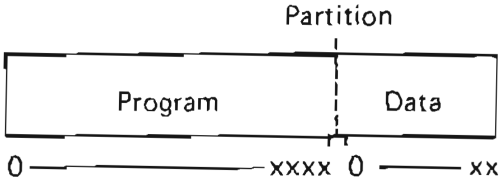
DATA STORAGE CAPABILITIES

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 in which register in data memory. If you are using very many data registers, though, you will probably need some form of bookkeeping to remember which registers contain what values.

SELECTION OF MEMORY SIZE (PARTITIONING)

The memory storage area of the calculator is designed to store both data and programs.



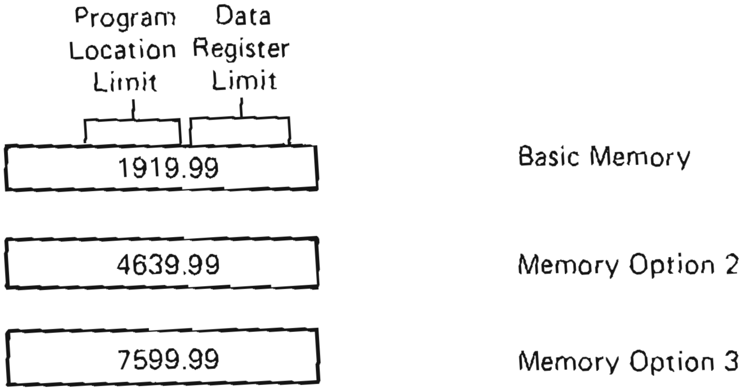
Memory Storage Area

The memory storage area of the basic SR-60A contains 340 registers of which 330 can be used for data storage. Initially 100 registers are allocated for the storage of data.

It is possible to greatly expand the memory capabilities of your calculator. Memory Option 2 provides 780 registers and Memory Option 3 boosts the capability to 1150 registers. See Appendix E for more details. Discussion here concerns the basic unit memory with the memory options immediately following in parentheses.

You can partition this storage area in increments of 10 registers, providing different ratios of program to data space to meet your needs.

Enter the number of sets of 10 registers you need for data storage and press **2nd X ≶ K**. For example, to obtain 100 data registers, press **10, 2nd X ≶ K** and the display shows the following



This shows that there are 100 registers (00-99) available for data storage and 1920 (4640 or 7600) set aside for program storage. A storage register can hold one data value or 8 program steps.

You can see from the first diagram that as you change the partition, you also move the data register numbers. If values have been stored, their register numbers will change when the partition is moved.

To check the current placement of the partition at any time, press **0, 2nd X ≶ K** and the existing partition is displayed in the above format. For more on partitioning, see *Storage Capacity and Partitioning* later in this section.

DATA REGISTER CONTROL

Because you can use up to 330 (770 or 990) data registers, you must specify which data register you are using by entering its address, N. If your partition is set for 10 data registers, N is a single digit, 0-9. For a partition of 10 to 100 registers, a two-digit address, 00-99, is needed. For more than 100 data registers, N requires 3 digits, 000-990.

Data Memory Keys

STORE N – **STORE** – Replaces the contents of R_N (data register N) with contents of the display register without affecting the display or pending operations. If the **STORE** key is pressed and followed by a nonnumerical key, the calculator will store the displayed quantity in R_0.

RECALL N – **RECALL** – Replaces the contents of the display register with a copy of R_N without affecting pending operations. If the **RECALL** key is pressed and followed by other than a nonnumeral key, the calculator will recall the contents of R_0.

EXCH N – **EXCHANGE** – Exchanges the contents of the display register with the contents of R_N without affecting pending operations. If the **EXCHANGE** key is pressed and followed by a nonnumeral key, the exchange operation is performed with R_0.

IV

Details

SUM N – SUM TO MEMORY – Algebraically adds the display register contents to the contents of R_N without affecting the display or pending operations. If the SUM key is pressed and followed by a nonnumeral key, the calculator sums to R_0.

2nd SUM N – SUBTRACT FROM MEMORY – Algebraically subtracts the display register contents from the contents of R_N without affecting the display or pending operations. If the keys **2nd, SUM** are followed by a nonnumeral key, the calculator subtracts from R_0.

PROD N – MULTIPLY INTO MEMORY – Multiplies the display register contents by the contents of R_N and stores the product in R_N without affecting the display or pending operations. If the **PROD** key is followed by a nonnumeral key, the calculator will multiply into R_0.

2nd PROD N – DIVIDE INTO MEMORY – Divides the display register contents into the contents of R_N and stores the quotient in R_N without affecting the display or pending operations. If the keys **2nd, PROD** are followed by a nonnumeral key, the calculator will divide into R_0.

CLEAR MEM CLEAR MEMORY – Sets the contents of all data registers to zero.

The results accumulated in the data registers are carried to 13 digits.

SHORT-FORM ADDRESSING

Addressing a particular register can be further simplified in some cases by eliminating leading zeros. The restriction on short-form addressing is that the entry following the single digit must not be a numeral key. When short-form addressing is used, the nonnumeral keystroke following the single-digit address first performs the pending data register operation and then performs the operation related to that keystroke. Short-form addressing should be used whenever possible, not only to save keystrokes, but to make programs compatible with any partition. The following example illustrates key sequences which are equivalent in the normal and short forms of address. Assume that the partition is for more than 100 data registers.

Normal Form	Short Form
RECALL 004	RECALL 4 =*
STORE 000	STORE +*
EXCH 000 + RECALL 010 =	EXCH + RECALL 10 =

*Or any nonnumeric key

Example: Solve the following problem using R_0 to avoid reentering the constant number.

$3.7 \times (2.14296)^3 + 9.7 \times (2.14296)^2 + 2.14296 = 83.1000007$

Press	Display
3.7 [×]	3.7
2.14296 [STORE] [y^x]	2.14296
3 [+]	36.41194836
9.7 [×] [RECALL] [x^2] [+]	80.9570407
[RECALL] [=]	83.1000007

Example: Use direct-register arithmetic to solve the following problem.

$$\frac{36 \times 3}{12} + \frac{54}{16-7} = 15$$

Press	Display	Register Contents R_{00}	 R_{01}
[CLEAR MEM] [CLEAR]	0.	0	0
36 [STORE] 00	36.	36.	0
3 [PROD] 00	3.	108.	0
12 [2nd] [PROD] 00	12.	9.	0
16 [STORE] 01	16.	9.	16.
7 [2nd] [SUM] 01	7.	9.	9.
54 [EXCH] 1	54.	9.	54.
[2nd] [PROD] 1	9.	9.	6.
[RECALL] 1	9.	9.	6.
[SUM]	6.	15.	6.
[RECALL] 00	15.	15.	6.

IV Details

LISTING DATA REGISTER CONTENTS

You may at any time obtain a partial or complete printout of the contents of the data registers.

2nd LIST **LIST DATA REGISTERS** – Prints out data register contents, beginning with the data register address indicated by the display. Press and hold the **HALT** key to stop the printout. Otherwise, the printer will automatically stop after printing out the last data register. The listing also halts with an overflowing register. Pressing the EE key prior to **2nd, LIST** will cause the contents of the data registers to be printed in scientific-notation format. The starting number in the display is used without regard to the decimal point. **FIX 9** should be pressed prior to listing.

```
Normal Data Register Listing      Data Register Listing in Scientific-Notation Format

                   0                                         0
         23.                                  2.3  01
                   1                                         1
 3.141592654                          3.141592654  00
                   2                                         2
 9.869604401                          9.869604401  00
                   3                                         3
 4.291132348                          4.291132348  00
                   4                                         4
 456678.2356                          4.566782356  05
                   5                                         5
         21.                                  2.1  01
                   6                                         6
  0.00000125                              1.25–06
                   7                                         7
          0.                                   0.  00
                   8                                         8
          0.                                   0.  00
                   9                                         9
          0.                                   0.  00
```

RECORDING AND READING DATA CARDS

The contents of the data registers can be recorded on a magnetic card for permanent storage. This feature is very convenient for applications such as inventory, payroll, or other situations where data needs to be permanently stored.

2nd WRITE **RECORD DATA CARD** – Instructs calculator to turn on card-unit drive motor to record data register contents on a magnetic card. The card will be recorded starting with the register number specified by the display register. **FIX 9** should also be pressed before recording data. Remember to place the black tab on the card first, then remove after writing.

2nd READ READ DATA CARD – Instructs the calculator to turn on card-unit drive motor to read data from a previously recorded magnetic card into the data registers. Key sequence should normally be preceded by the **CLEAR** key unless card is known to begin with other than R_0. In the latter case, enter starting data register number into the display before reading the card. Remember to press **FIX 9** before reading data.

A blank magnetic card has two sides, Side A and Side B. This means that one card can retain two independent sets of data. When you desire to record data on Side A, press **CLEAR, 2nd, WRITE** and insert the end of the card with the arrow corresponding to Side A pointed toward the slot in the calculator. Insert the opposite end to record on Side B. The contents of up to 60 data registers can be recorded on one side of a magnetic card. Recording all data register contents of an expanded-memory calculator requires several cards. In this case, record Side A as instructed in the following paragraph. To record data register contents beginning with R_{60}, enter **60** into the display, press **FIX 9, 2nd, WRITE** without pressing **CLEAR** and record Side B of card. Now the contents of registers 60-119 have been written. Press **120, 2nd, WRITE** to write out the next 60 registers, etc.

To record on a magnetic card, a black self-adhesive tab (furnished with your blank cards) must be placed over the square near the tip of the arrow corresponding to the side to be recorded. Press the **CLEAR, 2nd, WRITE** key sequence and insert the card into the calculator as you would a prerecorded program card. After recording, remove black tab to prevent accidental recording, and properly mark magnetic card for later reference.

CAUTION

Recording data on a magnetic card erases any data or program previously stored on the card. Do not use a card prerecorded by Texas Instruments for recording data register contents.

IV

Details

Packing Data Registers

Whenever a shortage of data registers develops, you can store two separate values in a single data register by separating the two with a decimal point. These situations can sometimes happen in programs that require a large number of data registers, so the program can combine two numbers using the following procedure.

Store the first number normally, then scale the next number so that it is now less than 1 and sum it into the same register. To retrieve the first number, recall the whole number and press the **Int x** key to discard the decimal portion. To retrieve the second number, recall the whole number and press **2nd Int x** to discard the integer portion and then scale the number to the proper magnitude.

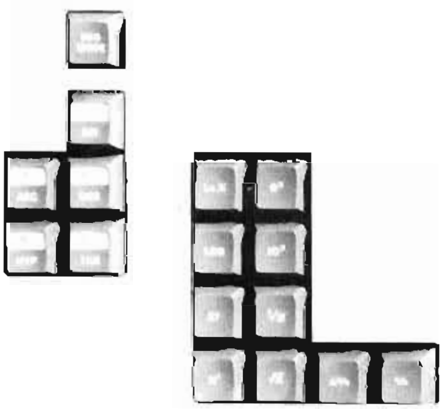
Example: Store and recall 1200 and 37 from data register 0.

Press	Display	Comments
1200 STORE 00	1200.	Store 1200 in R_0
37 ÷ 100 = SUM	0.37	Sum .37 into R_0
RECALL 00	1200.37	Recall R_0
Int x	1200.	First value recovered
RECALL 2nd Int x	0.37	Recall R_0, discard integer portion
× 100 =	37.	Rescale to proper magnitude

Note that you must know how many digits are in the second number so that you can select the proper scaling. Actually, more than two numbers can be stored in a register, but you must be careful when scaling them out.

ALGEBRAIC FUNCTIONS

There are 22 preprogrammed key functions which are first-level type hierarchy operations (except Δ%). This means that none of these operations will complete a pending operation. The Δ% key is an exception, because it can complete another Δ% operation and is designated a second-level operation. The 16 keys used for these functions are shown below. The special functions operate immediately on the contents of the display register.



Algebraic Function Keys

IV

Details

SQUARE AND SQUARE ROOT

x^2 SQUARE – Instructs the calculator to square the number in the display register.

Example: $4.2^2 = 17.64$

Press	Display
4.2 x^2	17.64

$\sqrt{x}$ SQUARE ROOT – Instructs the calculator to find the square root of the number in the display register. If x is negative, the calculator will find the square root of the magnitude of the number, but the display will flash.

Example: $\sqrt{6.25} = 2.5$

Press	Display
6.25 $\sqrt{x}$	2.5

The following example illustrates that first-level type operations will not affect pending operations.

Example: $\sqrt{19^2 + 13^2} = 23.02172887$

Press	Display	Comments
19 x^2 +	361.	
13 $\sqrt{x}$	3.605551275	$\sqrt{x}$ pressed by mistake
x^2	13.	Returns second entry
x^2	169.	Correct operation
=	530.	$19^2 + 13^2$
$\sqrt{x}$	23.02172887	

Details IV

RECIPROCAL

1/x RECIPROCAL — Instructs the calculator to find the reciprocal of the number in the display register. If the number is zero, the reciprocal will result in a flashing display.

Example: 1/3.2 = 0.3125

Press	Display
3.2 1/x	0.3125

PERCENTAGES

% PERCENT — Instructs the calculator to convert the number in the display register from a percentage to a decimal value (moves decimal two places to the left). When used after an addition or subtraction operation, add-on or discount calculations may be made. When used after other operations, percentages are converted to decimal values which become second operands.

Example: Add 6% to the amount $50.

Press	Display
CLEAR FIX 2	0
50 +	50.00
6 %	3.00
=	53.00

Example: Calculate the price of a $15.95 item that is discounted 20% and include a sales tax of 5%.

Press	Display
15.95 −	15.95
20 %	3.19
+	12.76
5 %	0.64
=	13.40

Example: What is 2.5% of $15?

Press	Display
15 ×	15.00
2.5 %	0.03
=	0.38

The above result is rounded to the nearest penny. To display the total result:

Press	Display
FIX 9	0.375

Example: $18 is 15% of what amount?

Press	Display
18 ÷	18.
15 %	0.15
=	120.

Notice that the solution to the last example would be the same for the problem, "What percentage of $15 is $18?". The answer to this is 120%.

Δ% PERCENT CHANGE – Instructs the calculator to determine the percentage change between two values. The first value is the data entry or calculated result being displayed when Δ% is pressed. The second value must be a data entry. Pressing the equals or another operation key completes the calculation and causes the percent change to be displayed.

Note: The percent change is calculated in the following manner:

$$\frac{(\text{second value} - \text{first value}) \times 100}{\text{first value}} = \text{percent change}$$

Example: A $5 item is marked down to $3. What is the discount percentage?

Press	Display
5 Δ%	5.
3 =	–40.

Example: A $3 item is marked up to $5. What is the markup percentage?

Press	Display
3 Δ%	3.
5 =	66.66666667

FACTORIAL

x! FACTORIAL – Instructs the calculator to find the factorial (1 X 2 X 3 X . . . X x) of the integer displayed. The largest integer for which the factorial may be calculated is 69. Negative numbers or decimal fraction numbers in the display or display register will cause a flashing display with question mark and the result will be the factorial of the integer part of the magnitude of the number.

Example: 7! = 5040

Press	Display
7 x!	5040.

LOGARITHMS AND ANTILOGARITHMS

Ln x NATURAL LOGARITHM – Instructs the calculator to find the natural logarithm (base e) of the number in the display register. Value of x must be greater than zero.

Example: Ln 5.4 = 1.686398954

Press	Display
5.4 Ln x	1.686398954

IV

Details

e^x e TO THE x POWER – Instructs the calculator to find the natural antilogarithm of the number in the display register.

Example: $e^{3.8} = 44.70118449$

Press	Display
3.8 e^x	44.70118449

LOG COMMON LOGARITHM – Instructs the calculator to find the common logarithm (base 10) of the number in the display register. Displayed value must be greater than zero.

Example: Log 1573 = 3.196728723

Press	Display
1573 LOG	3.196728723

10^x 10 TO THE x POWER – Instructs the calculator to find the common antilogarithm of the number in the display register.

Example: $10^{3.2} = 1584.893192$

Press	Display
3.2 10^x	1584.893192

TRIGONOMETRIC FUNCTIONS

Trigonometric values can be calculated for a wide range of positive or negative angles. As long as the trigonometric values are displayed in fixed-point format, all ten displayed digits are generally accurate for the range $-200\,\pi$ to $200\,\pi$ radians (-36000 to 36000 degrees). In general, the accuracy decreases one digit for each decade outside this range. Angles in the range of 10^{12} radians (10^{14} degrees) and larger are interpreted as zero radians or degrees.

The largest angular magnitude which will be returned by the calculator following an arc function is π radians (180 degrees). In addition, since the algebraic sign of a trigonometric result can indicate only two of the four quadrants, the arc-function results will be returned by the calculator as follows:

Arc Function	Quadrant of Resultant Angle
arc sine x	First (0° to 90° or 0 to $\pi/2$)
arc sine –x	Fourth (0° to –90° or 0 to $-\pi/2$)
arc cosine x	First (0° to 90° or 0 to $\pi/2$)
arc cosine –x	Second (90° to 180° or $\pi/2$ to π)
arc tangent x	First (0° to 90° or 0 to $\pi/2$)
arc tangent –x	Fourth (0° to –90° or 0 to $-\pi/2$)

where $x \geqslant 0$.

Note: Arc functions can also be written as $\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$, etc.

An example of this situation is that the sines of 30 degrees and 150 degrees are both 0.5 (a positive value). While the arc sine of 0.5 could be 30 or 150 degrees, the calculator will always return 30 degrees as the angle.

DEG MODE **DEGREE MODE** – Used to select the angular mode of operation for trigonometric functions, degrees/radians conversions and polar/rectangular conversions. When the indicator light above the **DEGREE MODE** key is off, the calculator assumes angular values to be in radians. When the indicator light is on, angular values are assumed to be in degrees. Pressing the **DEG MODE** key alternately selects degree and radian modes of operation. This operation may also be controlled by flag 7 in a program. See *Flag Operation* later in this section.

If it is necessary to convert an angular value from degrees to radians or radians to degrees, refer to *Angle Conversions* later in this section.

e_{11} SIN **SINE** – Instructs the calculator to find the sine of the angular value in the display register.

e_{7} ARC e_{11} SIN **ARC, SINE** – Instructs the calculator to find the arc sine ($\sin^{-1}$) of the number in the display register. Flashing display indicates the value is outside the required range –1 to +1.

Example: Sin 30° = 0.5

Press	Display
DEG MODE (Light on)	
30 e_{11} SIN	0.5

Example: Sin $\pi/3$ = 0.8660254038

Press	Display
DEG MODE (Light off)	
π ÷	3.141592654
3 =	1.047197551
e_{11} SIN	.8660254038

Example: Sin^{-1} .707 = 44.99134834°

Press	Display
DEG MODE (Light on)	
.707 e_7 ARC e_{11} SIN	44.99134834

e_{12} COS COSINE – Instructs the calculator to find the cosine of the angular value in the display register.

e_7 ARC e_{12} COS ARC, COSINE – Instructs the calculator to find the arc cosine ($\cos^{-1}$) of the number in the display register. Flashing display indicates the value is outside the required range –1 to +1.

Example: Cos 1.2 π = –0.8090169944

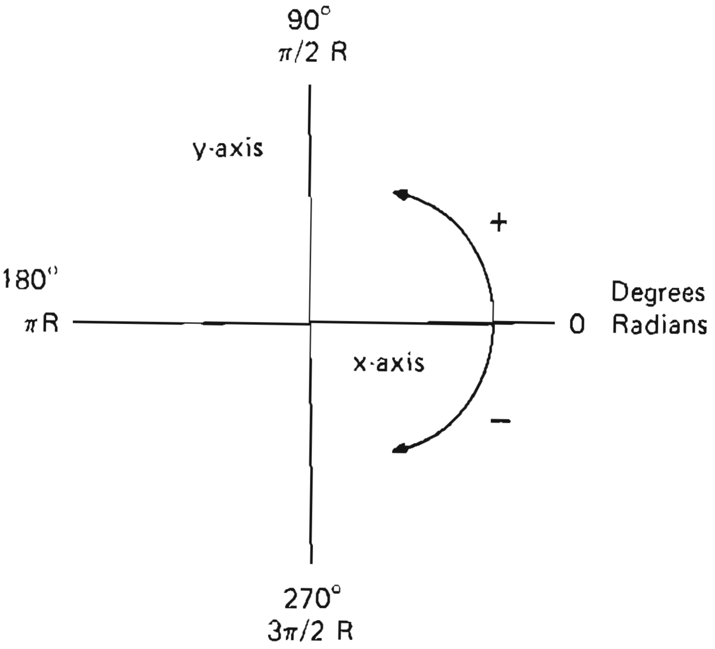
Press	Display
DEG MODE (Light off) 1.2 ×	1.2
e_{14} π =	3.769911184
e_{12} COS	–.8090169944

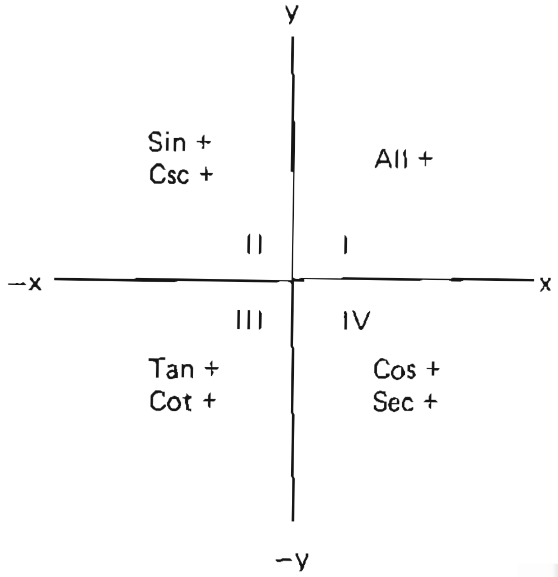
TAN TANGENT – Instructs the calculator to find the tangent of the angular value in the display register.

ARC **TAN** ARC, TANGENT – Instructs the calculator to find the arc tangent ($\tan^{-1}$) of the number in the display register.

Example: Tan 100° = –5.67128182

Press	Display
DEG MODE (Light on) **100** **TAN**	–5.67128182





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°.

The other trigonometric functions can be calculated as follows:

sec x = **COS, 1/x**
csc x = **SIN, 1/x**
cot x = **TAN, 1/x**

The same method can be applied to find the arc functions.

If your angle is expressed in degrees, minutes and seconds, the **D.MS** key can convert it to decimal form for you. See *Conversions* later in this section for all the details. Be sure your calculator is in the degree mode.

Example: sin 30° 13′ 48″ + tan 315° = −0.4965275891

Press	Display
30.1348 D.MS	30.23
SIN +	.5034724109
315 TAN	−1.
=	−.4965275891

HYPERBOLIC FUNCTIONS

HYP HYPERBOLIC FUNCTION – When used as a prefix to the **SIN**, **COS** or **TAN** keys, it instructs the calculator to find the selected hyperbolic function of the number in the display register.

Hyperbolic functions are functions of numbers, not angles, therefore, the status of the angular mode does not affect the hyperbolic functions. The key sequences for the hyperbolic functions are as follows:

Function	Key Sequence	Function	Key Sequence
sinh	HYP, SIN	$\sinh^{-1}$	ARC, HYP, SIN
cosh	HYP, COS	$\cosh^{-1}$	ARC, HYP, COS
tanh	HYP, TAN	$\tanh^{-1}$	ARC, HYP, TAN

When calculating inverse hyperbolic functions, the order of the prefixes may be **ARC, HYP** or **HYP, ARC.**

Example: Tanh 6.43 = 0.9999948

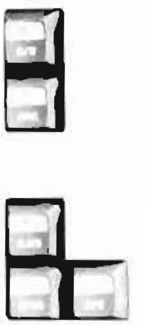
Press	Display
6.43 HYP TAN	0.9999948

Example: Sinh^{-1} 25 = 3.912422766

Press	Display
25 ARC HYP SIN	3.912422766

CONVERSIONS

The conversion functions are also first-level operations and will not affect pending operations. The keys for the preprogrammed conversions are shown below.



Conversion Keys

ANGULAR CONVERSIONS

e_6 D/R DEGREE/RADIAN CONVERSION – When the degree-mode indicator light is off, this key instructs the calculator to convert the angular value in the display register from degrees to radians. When the degree-mode indicator light is on, the conversion is from radians to degrees.

Example: Convert 30° to radians and the result back to degrees

Press	Display
30 e_6 D/R	.5235987756
DEG MODE (Light on)	.5235987756
e_6 D/R	30.

IV

Details

D.MS DEGREES-MINUTES-SECONDS – Instructs the calculator to convert the number in the display register from degrees-minutes-seconds to decimal degrees. The format for reentering degrees, minutes and seconds is DDD.MMSSsss, where:

DDD	is the degrees entry,
•	is the decimal point separating degrees and minutes,
MM	is the minutes entry (00 to 98),
SS	is the seconds entry (00 to 98), and
sss	is the decimal fraction of a second.

The D.MS function operates only with the number visibly displayed. Hidden guard digits or hidden digits as a result of using the **FIX** key are not used in the D.MS conversion.

ARC D.MS REVERSE DEGREES-MINUTES-SECONDS – Instructs the calculator to convert the number in the display register from decimal degrees to degrees-minutes-seconds. The format for the result is the same as for entering degrees-minutes-seconds in the other conversion: DDD.MMSSsss.

A number entered in the D.MS format is not limited to three digits for degrees or decimal fractions of a second. Four digits for the decimal fraction of a second may be entered if only two degree digits are needed and vice versa. The relationship of MM and SS to the decimal point is fixed. One minute must be entered as .01 or one second must be entered as .0001, etc. Results are positioned on the display to show the most significant digits.

Example: Convert 20° 97′ 25.4″ to decimal degrees and convert the result back to degrees-minutes-seconds again.

Press	**Display**
20.97254 D.MS	21.62372222
ARC D.MS	21.37254

The above result is interpreted as 21° 37′ 25.4″ which is not the same as the number first entered since the calculator automatically limits minutes and seconds in a result to 59.

The D.MS conversions are also useful in converting hours, minutes and seconds to decimal hours and vice versa. The format is almost identical: HHH.MMSSsss. This conversion simplifies addition or subtraction of numbers in hours-minutes-seconds format.

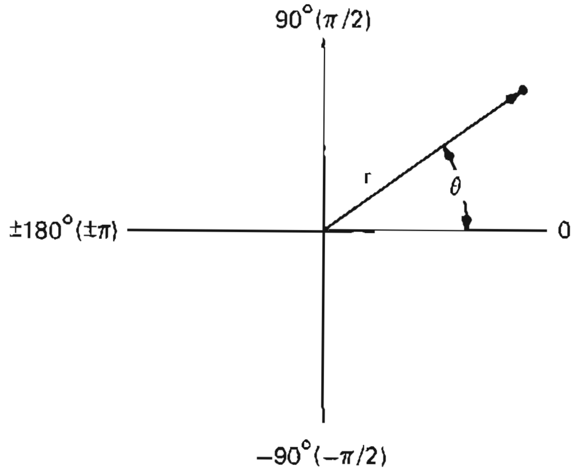
Example: Calculate the time difference between 2:45:38 and 6:15:21.

Press	Display	Comments
6.1521 e9 D.MS	6.255833333	Decimal hours
−		
2.4538 e9 D.MS	2.760555556	Decimal hours
=	3.495277778	Decimal difference
e7 ARC e9 D.MS	3.2943	3 hr, 29 min, 43 sec

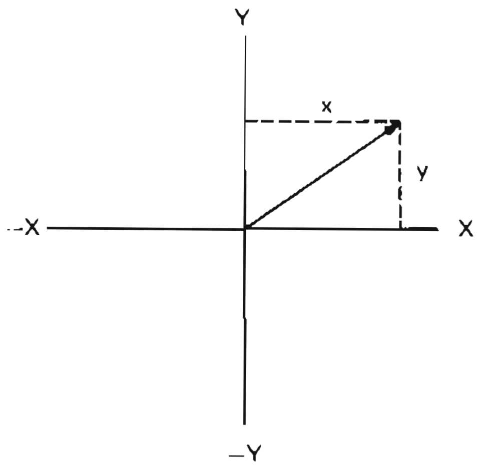
Note that solving time differences as indicated will **not** automatically compensate for times which are split by AM and PM time-change points unless you use military (24 hour) time, e.g. 1835 = 6:35 PM.

POLAR/RECTANGULAR CONVERSION

The polar/rectangular conversion instruction allows you to convert rectangular coordinates (x, y) to polar coordinates (r, θ) or vice versa. The following chart shows the variables for the two coordinate systems.



POLAR



RECTANGULAR

The **DEG MODE** key must be used to select degrees or radians to correspond with the entry or desired result form of angle θ. For polar-to-rectangular conversions, angle θ may be entered as a positive or negative angle. However, the calculator will return angle θ between 0 and ±180 degrees or 0 and ±π radians for rectangular-to-polar conversions.

IV

Details

e_{15} P/R **POLAR-TO-RECTANGULAR** – With the value for r in the R register and θ in the display register, this key converts the r, θ coordinates to x, y coordinates with y value displayed and x in the R register. The complete key sequence is as follows:

Enter r e_{10} x⇌R, Enter θ e_{15} P/R, display **y**, e_{10} x⇌R display x

e_7 ARC e_{15} P/R **RECTANGULAR-TO-POLAR** – With the value of x in the R register and y in the display register, this key sequence converts the x, y coordinates to r, θ coordinates with the θ value in the display and the r value in the R register. The complete key sequence is as follows:

Enter x e_{10} x⇌R, enter y e_7 ARC e_{15} P/R, display θ, e_{10} x⇌R display r

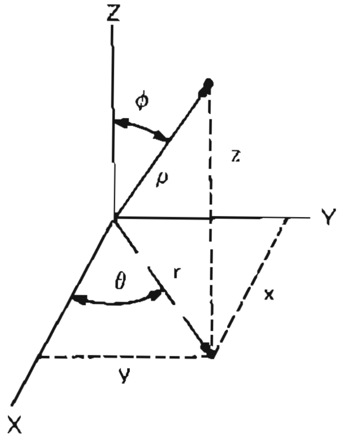
Example: Convert polar coordinates (5, 30°) to rectangular coordinates then convert back to polar with the angular result in radians.

Press	Display	Comments
DEG MODE (Light on)		
5 e_{10} x⇌R	–	See Note
30 e_{15} P/R	2.5	y value
e_{10} x⇌R	4.330127019	x value
DEG MODE e_{10} x⇌R e_7 ARC e_{15} P/R	.5235987756	θ in radians
e_{10} x⇌R	5.	radius

NOTE: The R register can only be cleared by pressing **0, X ≶ R,** or the **CLEAR ALL** key. Therefore, the last number entered as the result of a P/R conversion will appear when the **X ≷ R** key is pressed to enter new values.

SPHERICAL/RECTANGULAR CONVERSION

The **P/R** key can be used to convert from spherical coordinates (ρ, ϕ, θ) to rectangular coordinates (x, y, z) or vice versa. The conversion key sequences are based on the following diagram and equations.



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

$$y = \rho \sin\phi \sin \theta$$

$$x = \rho \cos \phi$$

$$\rho = \sqrt{x^2 + y^2 + z^2}$$

$$\phi = \cos^{-1} (z/\sqrt{x^2 + y^2 + z^2})$$

$$\theta = \tan^{-1} (y/x)$$

The key solutions for the three-dimensional conversions are:

Spherical to Rectangular

Press	Display
ρ e_{10} x⇌R	—
φ e_{15} P/R	r
e_{10} x⇌R	z*
θ e_{15} P/R	y
e_{10} x⇌R	x

Rectangular to Spherical

Press	Display
x e_{10} x⇌R	—
y e_7 ARC e_{15} P/R	θ*
z e_{10} x⇌R	r
e_7 ARC e_{15} P/R	φ
e_{10} x⇌R	ρ

*The z or θ values must be stored in a data register if they are needed after the conversion is completed.

IV

Details

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 **CLEAR ALL, 2nd, CLEAR MEM** 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 run 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.

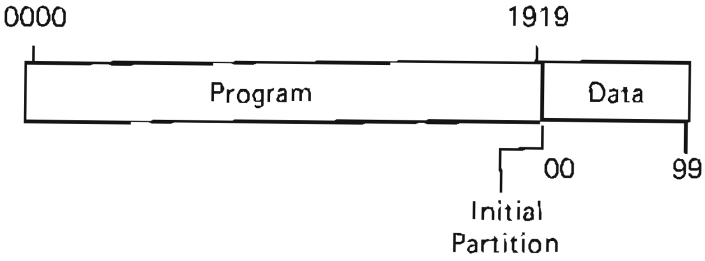
Details

IV

Storage Capacity and Partitioning

There is a memory storage area within your calculator. This area is used for both data storage and program storage. The discussion here is limited to the storage area of the basic calculator – 340 registers. See Appendix E for extended memory capabilities.

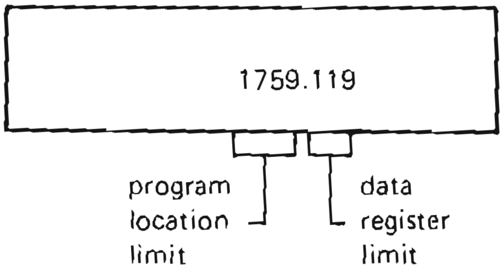
Each register can contain either one data number of up to 12 digits (and its exponent) or eight program instructions. You can partition the storage area to designate how many registers you need for data storage and how many for storing a program. When the calculator is first turned on, the storage area is partitioned to provide 100 registers (00-99) for data, leaving room for (340 – 100) X 8 = 1920 program instructions (0000-1919) as shown below.



Memory Storage Area

This area can be repartitioned at 10 register intervals, but you must have at least 10 registers each of program (80 steps) and data. For example, you can use 10 registers for data storage and (340 – 10) X 8 = 2640 locations for program instructions or 330 data registers and (340 – 330) X 8 = 80 program locations or any group of registers in multiples of 10.

To partition the storage area, enter the number of sets of 10 data registers you need, 1-33, and press **2nd**, X ≷ K. Remove all fix-decimal and scientific-notation formats before partitioning. For 120 data registers, press **12**, **2nd**, X ≷ K and the display is as follows:



This shows that there are 1760 program locations, 0000-1759, and 120 data registers, 000-119, available in the storage area. To check the current placement of the partition at any time, enter a negative number or zero and press **2nd**, X ≷ K and the existing partition is displayed in the format shown above.

NOTE

Whenever you move the partition, the position of all data registers changes because data register 0 is always the first register after program memory.

Basic Program Control Functions

The basic control functions of the calculator are those needed to allow entering a simple program. The figure below shows the basic program control keys. A few of these keys have more than one effect, depending on the circumstances. For this reason, some key definitions will appear again in this manual where they are critical to the operation of another function.

Basic Control Functions

LEARN – Pressing this key once puts the calculator in the learn mode of operation. The display is then broken into two fields: the program location number or address to the left and a key's alphanumeric key code to the right. Pressing **LEARN** again takes the calculator out of the learn mode and restores the display to its normal state.

Since the key code is limited to a maximum of three characters, it does not always read exactly the same as the key it represents. Refer to *Listing a Program* for the key code for each key as it is displayed or printed. These codes also apply to the printout during trace operation or when a program is listed.

RESET – Instructs the calculator to set the program pointer to zero. Also clears the subroutine return register and sets all flags to zero.

RUN – Instructs the calculator to run a stored program, beginning with the present position of the program pointer. When stored in a program, the run instruction has no effect except during single-stepping, when program execution resumes as if the **RUN** key were pressed.

HALT – Instructs the calculator to stop execution of a stored program and return operation to the keyboard. Pressing the **HALT** key will stop any type of operating program. However, the exact stopping address cannot be predetermined.

The **HALT** key should not be routinely used to manually stop a program with the intention to continue with the **RUN** key. If the program stops in the middle of a programmed number entry, the displayed part of the number is lost when the **RUN** key is pressed. When a program is stopped by pressing the **HALT** key, you should restart the program at a known point, such as at a label or the beginning of the program.

QUEUE – Pressed during operation from the keyboard, this key sets the program pointer to zero and automatically begins execution of the program. Pressing this key has the same effect as pressing **RESET**, **RUN**. This key is also used in prompting sequences as described in *Program Prompting Functions*.

GO TO – Instructs the calculator to set the program pointer to the address specified by the next key or key sequence. This is also called an unconditional transfer function.

The address may be specified as a digit or sequence of digits much as was done with the data register address operations. Depending on the placement of the partition, either two, three or four digits are required to identify program locations. The long form address scheme used with data memory is also applied to program storage.

IV

Details

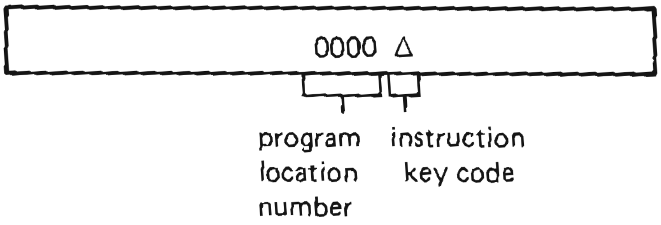
PAUSE PAUSE – Instructs the calculator to pause approximately 1/2 second and display a numerical result or alphanumeric message. Display time of a numerical result may be extended by repeating the pause instruction. A message is limited to a one time pause instruction immediately following the message field in the program. See *Alphanumeric Operations* for additional information.

2nd CLEAR MEM CLEAR PROGRAM MEMORY – Clears program memory, resets program location pointer to 0000, clears subroutine return register and sets all flags to zero. When encountered in a program, **2nd, CLEAR MEM** acts like a reset.

Short-form addressing is also possible for program operations. The leading zeros may be omitted if a nonnumeric key follows the address. **GO TO, 10, +** sets the instruction counter to 10. When the **GO TO** key is executed while sequencing through a stored program, the nonnumeric key is not executed since the program pointer is diverted before the + is encountered. However, when the **GO TO** key is pressed during keyboard operation and the short-form address is used, the nonnumeric key (+) is executed.

LEARN MODE

Once a calculation sequence has been determined, select the learn mode by pressing **2nd, CLEAR MEM, LEARN,** then key the sequence into program memory. **2nd, CLEAR MEM** assures that the program is keyed in beginning at location 0000. When you enter the learn mode, the display has the following format.



Program locations begin at 0000 and number consecutively up to your partition. Initially, program memory contains null instructions in all locations and these remain in all locations that are not deliberately changed.

Each location usually contains an operation or just a single digit.

An instruction in program memory is a three-character alphanumeric representation of the instruction itself. A null instruction is in each program location initially. These are replaced with program steps as the program is keyed in. The calculator normally indicates Δ in the instruction position when a program is being keyed in because the calculator automatically steps to the next available location as each step is keyed in.

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 or executed.

After keying a program into program memory, press **LEARN** 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 **RESET** or **2nd, CLEAR MEM**. Either sequence positions the program pointer to 0000, the first location of program memory. The **2nd, CLEAR MEM** sequence also clears program memory. The sequence **CLEAR ALL, NO** can be used, just be sure you want everything cleared.

2. Press **LEARN** to place the calculator in the learn mode. The special five-character display identifies this mode.

3. Key in your program completely: one step at a time, including all necessary **2nd** and **ARC** 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 available location is filled, the calculator automatically flashes the maximum program location and its corresponding program instruction.

5. Switch from the learn mode to keyboard control by pressing **LEARN** again. The display flickers while the label table(s) are being built.

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: Store a program that converts °F to °C. The equation to convert degrees Fahrenheit to degrees Celsius is

$$(°F) - 32 = \times 5 \div 9 = (°C)$$

Assume that the Fahrenheit value is in the display register when the program begins.

Press	Display	Comments
2nd CLEAR MEM	0	Clear program memory Resets program pointer to 0000
LEARN	0000 Δ	Enter learn mode
−	0001 Δ	Enter key sequence
3	0002 Δ	
2	0003 Δ	
=	0004 Δ	
×	0005 Δ	
5	0006 Δ	
÷	0007 Δ	
9	0008 Δ	
=	0009 Δ	
HALT	0010 Δ	Stops program and displays answer
LEARN	0	Exit learn mode

Now let's review the program in program memory

Press	Display	Comments
RESET	0	Reset program pointer to 0000
LEARN	0000 –	Minus stored in location 0000, etc.
STEP	0001 3	
STEP	0002 2	
STEP	0003 =	
STEP	0004 X	
STEP	0005 5	
STEP	0006 ÷	
STEP	0007 9	
STEP	0008 =	
STEP	0009 HALT	
STEP	0010 Δ	
LEARN	0	

RUNNING YOUR PROGRAM

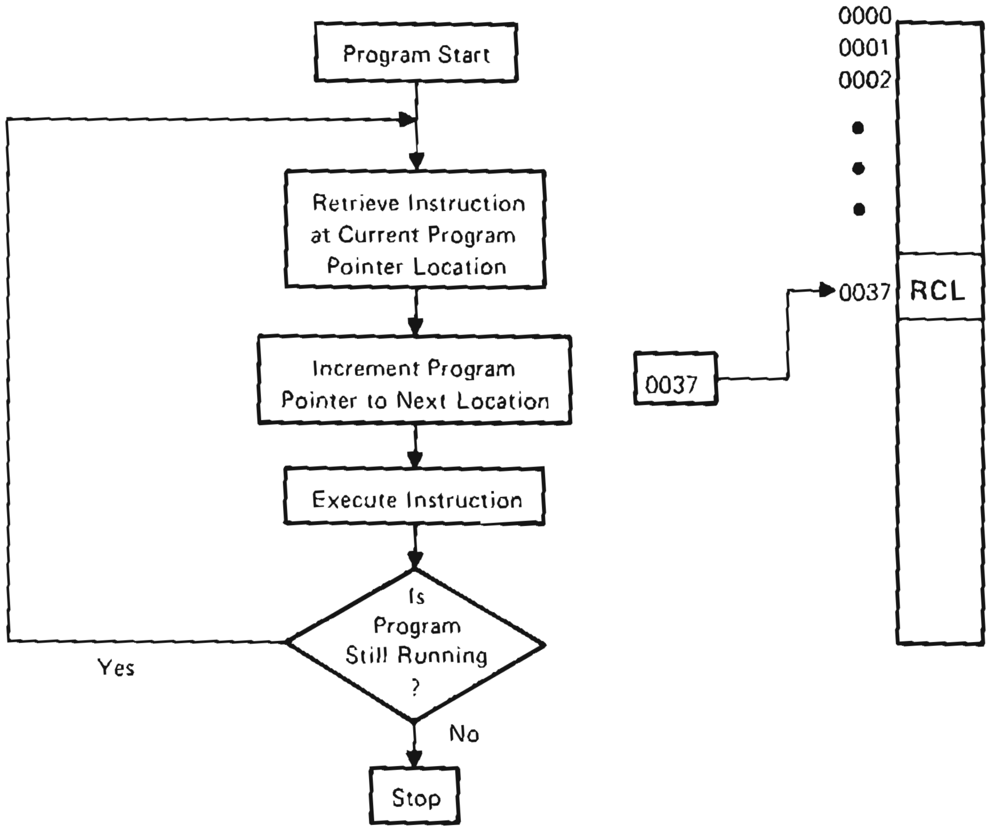
When running a program, the instructions are executed in sequential order beginning at the current location of the program pointer. (Advantageous exceptions will be discussed later.) To initiate the processing simply press the **RUN** key. The program pointer keeps up with exactly where processing is in the program. If the calculator attempts to execute past the program boundary, processing halts and the display flashes. Therefore, program should end with **HALT, RTN, GO TO** or **RESET.**

With the temperature conversion program still in program memory, convert 212°F and 32°F to Celsius.

Press	Display	Comments
212 RESET RUN	100.	212°F = 100°C
32 QUE	0.	32°F = 0°C

Notice that the display flickers while the calculator runs the program. The time required to run a program varies widely depending upon the type of program being run.

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 Start
Retrieve Instruction at Current Program Pointer Location
Increment Program Pointer to Next Location
Execute Instruction
Is Program Still Running ?
Yes
No
Stop
0000
0001
0002
0037
RCL
0037

Program Location Pointer

As additional programming capabilities are introduced, the role of the program location pointer will be expanded.

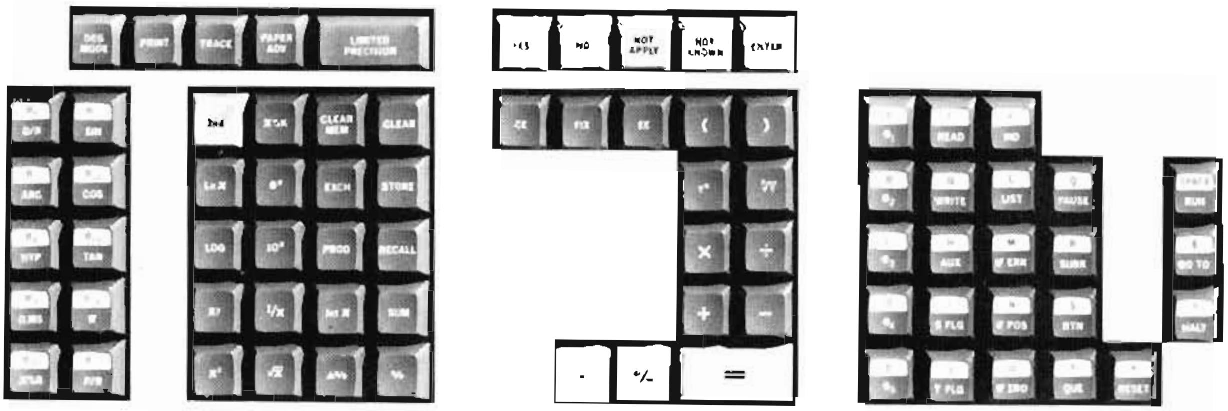
FUNCTIONS OF A LABEL

For programs that are very long or complicated, you can easily see that it is necessary to have reference points within the program. This is particularly true for branching and looping. Since every instruction has a specific address in program memory and since the program pointer can be arbitrarily set to any address, instruction addresses can be used as reference points. The use of a numerical address, however, is not always convenient. Your calculator provides a way to handle addressing without using the absolute numerical address. This method is by using labels.

Details IV

Labels

There are two types of labels in the SR-60A, *Program Labels and User-Defined Key Labels.* There are 77 primary labels possible on the calculator as shown below. Notice that the keys that are shaded may not be used as labels under normal circumstances.



Labels

P LABEL LABEL — When in the learn mode of operation, this key instructs the calculator to save the next key entry as a non-executable label. The non-executable label and the next address are saved in the primary label table. A key may only be identified as a label (prefixed with **LABEL**) one time in the program memory. However, a label may be called or transferred to as many times as desired.

Using labels in a program is similar to using tabs in a notebook. A label is a key that has been identified as a marker for a particular segment of a program (sometimes identified as a subroutine) or even a complete program. When a label is called in a program or from the keyboard (GO TO X^2, for example), the calculator will find the label in the label table and set the program pointer to the corresponding address. There are two major advantages in using labels. One is that labels may be used in place of the three- or four-digit addresses required for conditional and unconditional program transfers. The second is that exclusive use of labels in a program permits inserting or deleting program steps without worrying about the changing addresses of other instructions.

IV

Details

Any of the 77 possible label keys can be used as program labels. However, it is normally desirable to reserve the 15 keys identified with e_1 through e_{15} as user-defined key labels. The main difference between the two label types is the ways they are used. Program labels are used internally in a program to identify required transfer points for subroutines, if-conditions, etc. User-defined key labels are used when easy access to specific parts of a program is needed from the keyboard. See *User-Defined Keys* description below.

As the program is entered, the calculator develops a label table. The address of the first executable instruction in a labeled sequence is saved in the label table. For example, consider the following program segment:

Location	Key
0101	LABEL
0102	X^2
0103	2
0104	+
0105	2
0106	=

The non-executable sequence LABEL, X^2 alerts the calculator to save the address 0103 in the label table as being associated with the label X^2. Consequently, when an instruction such as GO TO, X^2 is encountered, the program pointer is set to 0103.

Even though X^2 has been used as a label, this will not prohibit its being used elsewhere in the program in its normal role. By using the label table, the calculator is able to determine from context which usage is intended.

User-Defined Keys

There are five keys, e_1 through e_6, which are specifically designated as user-defined keys, and ten more (e_6 through e_{15}) which have the option of being designated as user-defined keys. When any of the first five keys are prefixed with the LABEL key in a program and the same key is pressed on the keyboard, the calculator will retrieve the address at the label from the label table and automatically begin program execution starting at that address. The other ten keys will behave in the same manner if they are prefixed with the **2nd** key.

It is important to note that the e_6 through e_{15} identifiers never appear in the program memory or on a program listing. The actual label for e_6 is D/R, e_7 is ARC, etc. Even though you desire to use e_6 as a user-defined label, you need only to store LABEL, D/R in the program. The **2nd** key prefix when calling the label is what makes it function as a user-defined key.

Another unique feature of the user-defined keys is that when stored in a program, the calculator will treat the stored instruction the same as when you manually press the key, with one addition. When a user-defined key is stored alone in a program (not prefixed with **GO TO**, etc.), the program segment is treated as a subroutine and operates with the same rules set up when actually using the **SUBR** key. The description of *Subroutine Transfers* further illustrates this function.

To illustrate user-defined keys and types of labels, consider a program to cube a number and add the result to 27. That is, calculate $y^3 + 27$ for various values of y. Three versions of the program are shown to compare definition and usage: using + as a label, e_1 as a label and e_6 as a label.

Location	Key	Comments
0000	LABEL	
0001	+	Label
0002	y^x	These two instructions cause whatever is in the display to be cubed
0003	3	
0004	+	This is the normal usage +
0005	2	27
0006	7	
0007	=	
0008	RTN	Either a **HALT** or **RTN** can be used here

It is probably better to use **RTN** if this segment of code is ever to be used as part (subroutine) of another program. If the program is a stand-alone program (not a subroutine), **RTN** acts like **HALT**.

To enter the program into program memory, press **RESET**, **LEARN**, then enter in sequence the above segment of code. To take the calculator out of learn mode, press **LEARN** again. To execute the program just entered, press **RESET** , key in a value for y and then press **GO TO**, +, **RUN** and the answer will be displayed.

Example:

Press	Display	Comments
Y RESET	0	
3 $ GO TO + SPACE RUN	54.	**RUN** tells the calculator to start the calculation

Now modify the already stored program to relabel this program e_1. Press **RESET**, **LEARN**, the display will read "0000 LBL". Press **STEP** (the single-step key). The display now reads "0001 +". Now press e_1. The + which was at location 0001 has now been replaced by e1. To see this, press **BSTEP** and the display reads "0001 e1". Press **LEARN** to return to keyboard control.

Press	Display	Comments
3 e_1	54.	e_1 behaves like a function key such as e^x or **ln x**, etc.

Notice that there is no need to press RESET or GO TO before e_1, nor is there any need to press RUN. By way of comparison perform the following:

Press	Display	Comments
3 GO TO e_1	3	The program pointer is set to e_1.
RUN	54.	Program is executed.

In the same way as before, press RESET, LEARN, STEP the display shows "0001 e1". Replace e_1 in this instruction by e_6. This is done by pressing D/R. When there is no danger of confusion, this will be denoted simply by e_6. Now press BSTEP. The display reads "0001 D/R". The designation D/R is the label name for e_6. Return the calculator to calculate mode by pressing LEARN.

Press	Display	Comments
3 2nd D/R	54.	Thus to execute the program, the second function of D/R is needed.

Of course, the same principles are followed for e_7 through e_{15}.

Secondary Labels

LABEL • SECONDARY LABELS – Operates the same as above, except that a second label table contains these labels, thus almost doubling the number of labels available. There are only 76 secondary labels since • is used to signal a secondary label.

All the labels previously defined are called *primary labels*. These keys can also be used as secondary labels if preceded by a decimal point. Of course, • cannot now be used as a label because it is the secondary label indicator. However, label • is still a valid primary label if no secondary labels are requested. To enter primary label X^2, key in **LABEL**, X^2 as before. To enter secondary label X^2, press LABEL, •, X^2. To prepare the calculator for a secondary set of labels, **2nd**, LABEL must be pressed before entering the learn mode. When you exit the learn mode a secondary label table is constructed in addition to the one for primary labels.

Programs from magnetic cards can still be read as before. If only primary labels are used, the program is used normally. If secondary labels are used, a secondary label table must be constructed. In this case, press HALT after the last card has been read, before the program starts to run, and press **2nd**, LABEL, **LEARN**, **LEARN** to build the table. Now press QUE to restart the program. An alternate method is to press **2nd**, **CLEAR** MEM, RESET, **2nd**, LABEL, READ, READ, . . ., READ. After the last read, both label tables will be generated and you can run the program normally.

IV

Details

EDITING

In the previous examples, two keys were used to assist in altering or editing the program, STEP and BSTEP. There are four of these editing keys. They affect program storage or the program pointer; they may not be written into a program. The two keys already considered allow stepping forward and backward through the storage locations to verify or change program steps. The following figure illustrates the editing keys and the printer controls that are also very useful in editing a program.

Edit Keys

STEP – Causes the program location pointer to be incremented by one. In the learn mode, this causes the next storage location to be displayed. While not in the learn mode, pressing this key causes the program to be executed one step at a time except an alphanumeric field will run automatically and the STEP key must be held down to stop the program at the end of the field.

BACK STEP – Causes the program pointer to be decremented by one. In the learn mode, this causes the previous instruction and address to be displayed. While not in the learn mode, pressing this key has no effect.

INSERT – In the learn mode, this key causes the instruction at the displayed address and all following instructions to shift to the next higher address. A null instruction is inserted at the displayed address. The program instruction in the last available location is lost when the INSERT key is pressed.

DELETE – In the learn mode, this key causes the instruction at the displayed address to be deleted and shifts all the following instructions to the next lower numbered address. A null instruction is placed in the last program memory location.

Details

As you have seen, program storage may be examined by pressing **RESET** and **LEARN** and then stepping through a program using the **STEP** key. The program pointer has an associated alphanumeric key code which may be used to verify the stored program. A stored key instruction can be replaced by pressing another key when the unwanted key code is displayed. The change may be verified using the **BSTEP** key. The **INSRT** key permits adding a new key instruction at any point in a program without affecting existing instructions.

Note: If these keys don't seem to work in the learn mode, you're probably in alpha entry mode. Press **ALPHA** and try the keys again.

Return to the previous program example with e_1 as the label.

Location	Key Code
0000	LBL
0001	e_1
0002	y^x
0003	3
0004	+
0005	2
0006	7
0007	=
0008	RTN

Suppose after reviewing the program you decide to insert or delete steps rather than just replace a step. The following example shows one way to modify the stored program to calculate $y^{3.5} + 7$ for various values of y.

Press	Display	Comments
GO TO e_i LEARN	0002 y^x	
STEP	0003 3	
STEP	0004 +	
INSRT	0004 △	Inserts null instruction at 0004 moves + to 0005
INSRT	0004 △	Inserts null instruction at 0005 also. Moves + to 0006. Now have two blank spaces in which to insert • and 5.
•	0005 △	Still a null instruction at 0005
5	0006 +	
DSTEP	0005 5	Thus .5 has been inserted
DSTEP	0004 •	
STEP	0005 5	
STEP	0006 +	
STEP	0007 2	The 2 is an unwanted step
DLETE	0007 7	2 has now been deleted
STEP	0008 =	
STEP	0009 RTN	
LEARN	0	

Example: What is the value of the expression $y^{3.6} + 7$ with y = 6, .59 X 10^5, –27

Enter	Press	Display	Comments
6	A e_1	536.0897844	
.59	EE	.59 00	
5	A e_1	4.988638576 16	
27	+/– A e_1	1.022828681 05?	Flashing display

As noted, the last result shows a flashing display which indicates some invalid operation. In this particular case the invalid statement is that the –27 entry was used in the program as a y value in a y^x function. The calculator provided an answer by using 27 as a positive number, but the display flashed with a question mark to indicate that the entry was not used exactly as entered. Also note that the last result was displayed in scientific-notation format because neither **CLEAR** nor **2nd**, **EE** was used after the preceding calculation.

LISTING A PROGRAM

After a program is keyed into the calculator program memory, the convenience of listing the program contents with the printer is an asset to checking and editing a program.

L LIST PROGRAM LIST – Instructs the calculator to sequentially print the contents of the program memory beginning with the current address in the program location pointer. The printed listing shows the location numbers and the corresponding instruction or key code stored at the location.

When a program listing is to start at location 0000, press **RESET**, **LIST**. If the listing is to start at any other location, press **GO TO**, the location number, **LIST**. A program listing will continue until a null instruction is encountered in the program or until the **HALT** key is pressed and momentarily held down. Note that when a listing is stopped by a null instruction, the calculator remains in the learn mode of operation so that any necessary program alterations can be made. Therefore, the LEARN key must be pressed to continue normal keyboard operations.

The key code for most keys is the same as the symbolization on the respective key. Since key codes are allowed a maximum of three characters when displayed or printed, some key codes differ from the keys. The following list shows these key codes and the keys they represent.

Key Code	Key	Key Code	Key
ALF	ALPHA	PRE	LIMITED PRECISION
CLR	CLEAR	PRT	PRINT
CMS	CLEAR MEM	RCL	RECALL
DEG	DEG MODE	RD	READ
ENT	ENTER	RST	RESET
GTO	GO TO	SBR	SUBR
HLT	HALT	SF	S FLG
IFE	IF ERR	STO	STORE
IF+	IF POS	TFS	T FLG
IFZ	IF ZRO	TRC	TRACE
INT	Int x	UNK	UNKNOWN
LBL	LABEL	WRT	WRITE
LST	LIST	XM	EXCH
N/A	NOT APPLY	Π	PROD
PA	PAPER ADV	Σ	SUM
PAU	PAUSE	II	2nd

Printer Key Codes

Example: Obtain a listing of the last program entered which performs the calculation $y^{3.5} + 7$ when e_1 is pressed.

Press **RESET, LIST:**

Print

```
0000   LBL
0001   e1
0002   Yx
0003   3
0004   .
0005   5
0006   +
0007   7
0008   =
0009   RTN
```

Alphanumerics in a program print the actual alphanumeric symbol you wish to display or print. When **ALF** is encountered in a program, all characters until the next **ALF** are interpreted as alphanumerics. Enter the learn mode and key in **LABEL, e$_1$, ALF, S, R, – , 6, O, A, ALF**. Exit the learn mode and press **RESET, LIST** and the following listing results.

```
Print

0000  LBL
0001  e1
0002  ALF
0003   S
0004   R
0005   –
0006   6
0007   O
0008   A
0009  ALF
```

Listing a program should never begin inside an alphanumeric field, because the calculator won't interpret alphanumeric or non-alphanumeric information correctly.

PROGRAM TRACE OPERATIONS

The trace operation was briefly described earlier to print manual calculations. There are no additional functions of the **TRACE** key, however, the trace operation becomes a very useful tool when a program does not run properly. When a program is run with the trace operation selected, the printer will produce a sequential record for each instruction performed by the program. The printed record allows you to analyze the program to determine where an error may have occurred.

The functions executed in a program are identified on the right side of the printout, using the same key codes identified in the previous description of listing a program.

Example: Reenter the previously listed program and run with the trace operation selected, using **6** as the entered number.

Press	Print	
CLEAR		
TRACE (light on)	0	
6 A		e1
	6.	Y^x
	3.5	+
	7.	=
	536.0897844	
	536.0897844	
TRACE (light off)	536.0897844	TRC

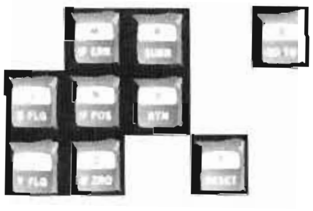
It is important to note that **TRACE** is also controlled by flag 9 and thus pressing **QUE**, **RESET**, or **2nd**, **S FLG 9** will turn the trace operation off. The latter two operations executed in a program will also turn trace off. For example, a program which automatically starts over by using **RESET** will automatically turn off trace when the reset instruction is executed.

The trace operation is relatively simple, however, you should practice using the trace operation with programs you understand to become familiar with the trace printing format. The trace operation is also illustrated in *Basic Prompting Examples.*

PROGRAM TRANSFER OPERATIONS

The normal order of program execution is one step at a time with the program pointer incrementing by one. A *program transfer operation* causes the program pointer to be set to a location other than the next sequential location. When a transfer occurs, the program continues running from the new location.

Program transfer operations are divided into two major groups: conditional and unconditional. A *conditional transfer operation* causes the calculator to make a decision based on a specified condition. A transfer is then made depending on whether the condition is true or false. For example, suppose you want the calculator to branch if the number in the display is positive. The calculator can be programmed to examine the display and decide if it is positive. If the display is positive a transfer will take place. If the display is not positive, no transfer will occur. An *unconditional transfer operation* causes the transfer to occur without regard to display or any other conditions. The following figure shows the transfer keys.



Conditional and Unconditional Transfers

Unconditional Program Transfers

There are three basic types of unconditional program transfers:

1. User-defined key instructions e_1 through e_5 and **2nd**, e_6 through **2nd**, e_{15}
2. The GO TO instruction
3. The subroutine instruction SUBR

The user-defined keys are unique in operation. They can operate as normal program labels when used as part of another program transfer instruction, otherwise, they function like the subroutine instruction. Program transfers with user-defined labels are shown with the subroutine examples.

$ GO TO yyyy or label — GO TO — Sets the program location pointer to specific location (yyyy) or label. When encountered in a program, this instruction is performed without interrupting program execution. Otherwise, press RUN to execute program or STEP to single-step the program. Note that preceding a user-defined key with GO TO on the keyboard instructs the calculator to go to the label but not to execute the program. In a program, the same instruction sequence transfers the program to specified location (yyyy) or label without being treated as a subroutine.

Go To Location (yyyy)

Location	Key Code
0021	=
0022	GTO
0023	5
0024	5
0025	LBL
0054	X
0055	RCL
0056	0
0057	8

Location transfer

Go To Label

Location	Key Code
0101	GTO
0102	RUN
0103	2nd
0104	IF+
0105	e_1
0150	+
0151	LBL
0152	RUN
0153	y^x

Label transfer

In addition to the GO TO key, the subroutine key acts as an unconditional transfer instruction. It is used to transfer program control to a segment of the program defined as a subroutine, execute the subroutine and automatically return to and continue the main program. The methods of controlling the subroutine instruction are the same as for the GO TO instruction.

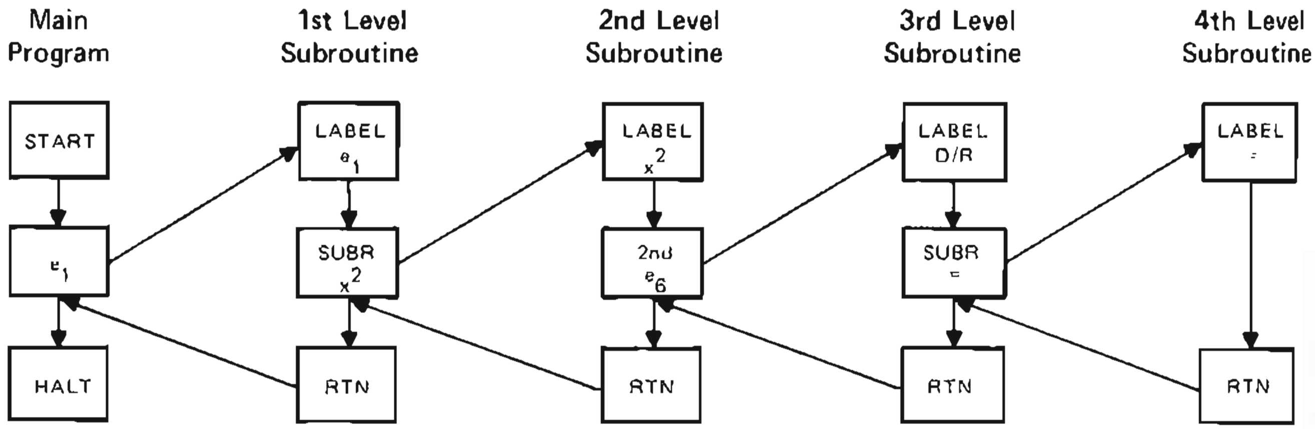
R SUBR **yyyy or label** – SUBROUTINE – Instructs the calculator to set the program pointer to a specified program location (yyyy) or label. From the keyboard, the subroutine key operates identically to the GO TO key. In a program, the calculator stores the address of the next instruction following **SUBR**, yyyy (or label) in the subroutine return register. The calculator then transfers to and executes the subroutine program and automatically returns to the last address stored in the subroutine return register when a return instruction (RTN) is met. The subroutine return register will hold up to 12 return addresses at a time. The addresses are retrieved on a last in, first out basis.

S RTN RETURN – Causes the program pointer to return to the last address placed in the subroutine return register by a subroutine or user-defined key instruction. If more than 12 subroutine levels are called, the RETURN instruction will cause the program pointer to return to the eleventh subroutine level as if the higher level subroutines did not exist. If the RETURN instruction is executed in a program when no address is in the subroutine return register, the program will halt as if the **HALT** instruction had been encountered and return operation to the keyboard. Using a 13th subroutine level can be used to create a loop.

Y RESET RESET – Instructs the calculator to set the program location pointer to zero and to clear all addrèsses from the subroutine return register. Also sets all flags to zero.

Important subroutine features to remember are:

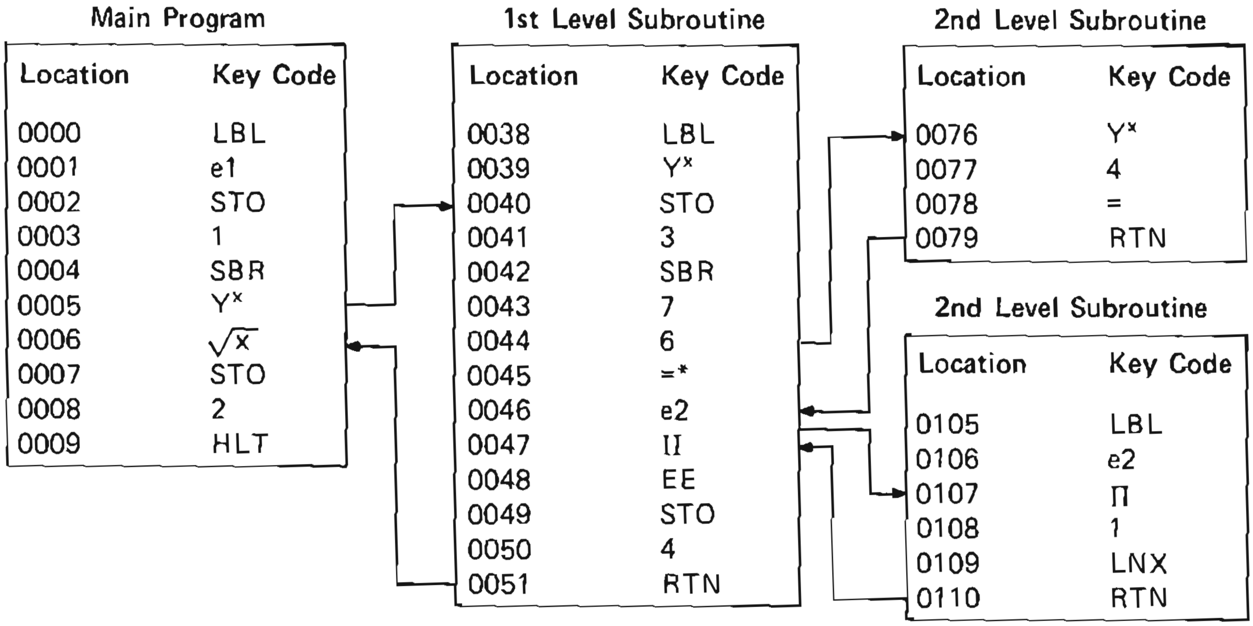
1. Twelve return addresses can be stored in the subroutine return register at a time.
2. Subroutines should always be terminated with a return instruction (RTN).
3. When not used in conjunction with a transfer instruction, a user-defined key will cause a transfer identical to the subroutine instruction.
4. The subroutine return register operates on a last-in, first-out basis. The return instruction retrieves the addresses (yyyy) one at a time. If there is no address in the subroutine return register, program execution will stop at the return instruction.
5. **RESET, CLEAR ALL, QUE** and **2nd, CLEAR MEM** clear all addresses from the subroutine return register.



Four-Level Subroutine Flow Diagram

If at some time it is necessary to use a halt instruction in a subroutine to stop the program, there are three alternatives you may use. The first is to view the intermediate result or enter data and press RUN to continue the program. The second is to press RESET before running any other part of the program. The third is to include a print instruction in the program for the result, followed by a reset instruction. The third alternative is the most desirable in prompting programs since it will restart the program as well as clear the subroutine return register. A simplified flow diagram for four subroutine levels is shown below.

When executed in a program, the subroutine instruction will terminate a number entry. Thus, if a 2 is in the location immediately preceding SBR and a 5 is in the first location of the subroutine, the 2 is lost and only the 5 is taken as the number entry. A return instruction at the end of a subroutine does not terminate a number entry unless it stops the program.



Main Program

Location	Key Code
0000	LBL
0001	e1
0002	STO
0003	1
0004	SBR
0005	Y^x
0006	$\sqrt{x}$
0007	STO
0008	2
0009	HLT

1st Level Subroutine

Location	Key Code
0038	LBL
0039	Y^x
0040	STO
0041	3
0042	SBR
0043	7
0044	6
0045	=*
0046	e2
0047	II
0048	EE
0049	STO
0050	4
0051	RTN

2nd Level Subroutine

Location	Key Code
0076	Y^x
0077	4
0078	=
0079	RTN

2nd Level Subroutine

Location	Key Code
0105	LBL
0106	e2
0107	Π
0108	1
0109	LNX
0110	RTN

*Care must be taken when using short-form addressing to call a subroutine. Unless some nonnumeric key is inserted between 6 and e2, e2 will be skipped.

Two-Level Subroutine Example

Conditional Program Transfer (Branching)

There are three basic types of conditional transfer instructions. One is an instruction which makes a decision based on the contents of the display register at the time of the instruction. The second is an instruction which makes a decision based on the status of one of ten flags. The third uses the contents of data register 0 to make its decision. The general operation of a conditional transfer instruction is to test or ask a true-false question about the status of a data register, the display register or flag. The calculator then determines whether the answer is true or false. If true, a program transfer occurs to the location (yyyy) or label immediately following the instruction. If false, the location (yyyy) or label is skipped and program execution continues with the next instruction. The **2nd** key may be used with a conditional transfer instruction to reverse the sense of the test or question and the indirect **IND** key may be used for indirectly transferring to a program location when the answer is true.

IMPORTANT: A conditional transfer to a program location (yyyy) or label is a one-way transfer (an implied GO TO) and will not automatically return even if the transfer is to a label that is a user-defined key.

If-Condition Instructions

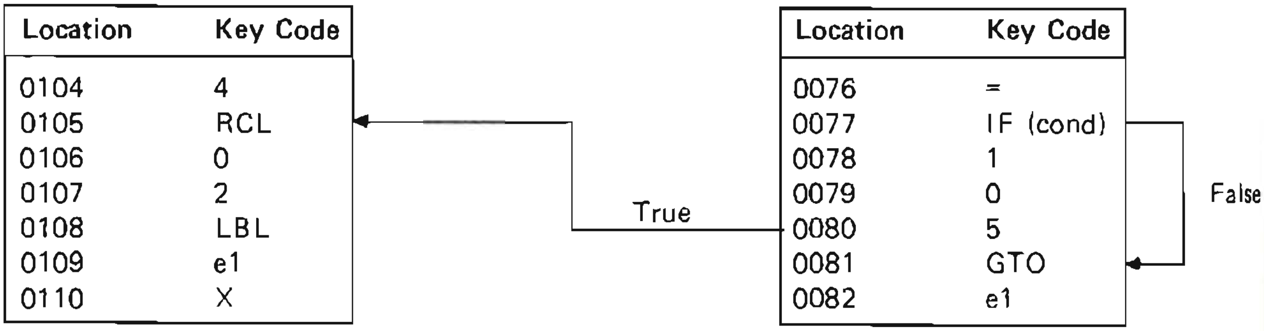
To permit easy interpretation of program execution, the following descriptions are written as if the program were being executed one step at a time (as when using STEP). The effect is equivalent to normal program execution.

0 IF ZRO yyyy or label — IF ZERO — If the number in the display is zero, the program location pointer is set to the specified location (yyyy) or label. If the number in the display register is not zero, program execution continues with the instruction immediately following the specified location (yyyy) or label.

CAUTION: The number displayed and the number in the display register are not always the same. For example, with **FIX 2** selected, 1.001 in the display register will be displayed as 1.00 since the display is rounded to two decimal places. Also, just because two functions are mathematically equal, do not assume that their calculated difference is equal to zero. See *Displayed Results versus Accuracy* for more information.

2nd 0 IF ZRO yyyy or label — IF NOT ZERO — Reverses the sense of the If Zero instruction. The transfer occurs if the display is not zero.

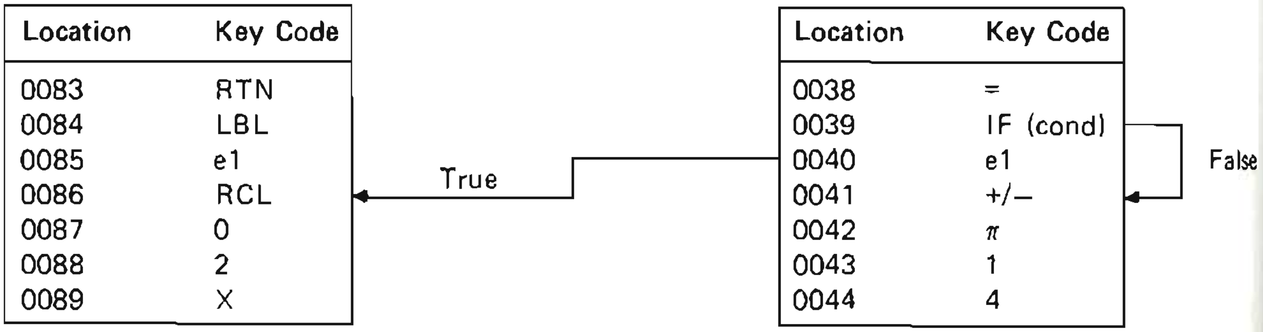
N IF POS yyyy or label — IF POSITIVE — If the number in the display is zero or positive, the program pointer is set to the specified location (yyyy) or label. If the displayed number is negative, program execution continues with the instruction immediately following the specified location (yyyy) or label.



Location	Key Code
0104	4
0105	RCL
0106	0
0107	2
0108	LBL
0109	e1
0110	X

Location	Key Code
0076	=
0077	IF (cond)
0078	1
0079	0
0080	5
0081	GTO
0082	e1

If-Condition Location Transfer



Location	Key Code
0083	RTN
0084	LBL
0085	e1
0086	RCL
0087	0
0088	2
0089	X

Location	Key Code
0038	=
0039	IF (cond)
0040	e1
0041	+/–
0042	π
0043	1
0044	4

If-Condition Label Transfer

2nd IF POS yyyy or label – IF NEGATIVE – Reverses the sense of the if-positive instruction. The transfer occurs if the displayed number is negative.

IF ERR yyyy or label – IF ERROR – If the current value in the display register is flashing (denoting overflow, underflow, or invalid operation), the program pointer is set to the specified location (yyyy) or label. If the display is not flashing, program execution continues with the instruction immediately following the specified location (yyyy) or label.

2nd IF ERR yyyy or label – IF NO ERROR – Reverses the sense of the if-error instruction. The transfer occurs if the display is not flashing.

An instruction or result which normally causes a flashing display will not interfere with execution of a program unless programmed to do so with the if-error instruction. For invalid operations not related to a displayed result, press **CE** to stop flashing display without affecting the displayed number.

Expanding If-Condition Instructions

When used directly, the if-condition instructions described involve only one number or result to make a transfer decision. It is desirable at times to make a transfer decision based on two numbers or results. The table below illustrates the key sequences that may be used to perform transfer decisions involving two numbers. Notice that the entry sequence of the two numbers is important in determining the decision to be made. In any case, a transfer to location yyyy or label occurs if and only if the test is true.

DECISION		KEY SEQUENCE
If a = b		a − b = IF ZRO yyyy or label
	or	b − a = IF ZRO yyyy or label
If a ≠ b		a − b = 2nd IF ZRO yyyy or label
	or	b − a = 2nd IF ZRO yyyy or label
If a > b		b − a = 2nd IF POS yyyy or label
If a ⩾ b		a − b = IF POS yyyy or label
If a < b		a − b = 2nd IF POS yyyy or label
If a ⩽ b		b − a = IF POS yyyy or label

Transfer Decisions with Two Numbers

It is possible to perform the mathematical operation a − b = without affecting previously entered calculations which are pending. The two key sequences which will do this are:

(a − b)

or

a STORE N b 2nd SUM N RECALL N

Flag Operation

Other conditional transfer instructions cause transfers because of predefined operations (positive, negative, zero, etc.). Flags permit you to define the condition under which a transfer takes place by setting a flag for later reference. Using flags, you have the option of defining ten different cases for conditional transfer in addition to the ones already described.

S FLG z – **SET FLAG** – Instructs the calculator to set (raise) the flag specified (z). Up to ten flags may be independently set (z = 0 through 9).

2nd S FLG z – **RESET FLAG** – Instructs the calculator to reset (lower) the flag specified (z). Up to ten flags may be independently reset (z = 0 through 9).

RESET **RESET** – Instructs the calculator to set the instruction counter to zero, clear all addresses from the subroutine return register and reset all flags. This occurs both from the keyboard and in a program.

Flags may be set or reset manually from the keyboard prior to executing a program and set or reset according to instructions stored in a program. The test flag instruction is used to test the condition of a specific flag.

T FLG **z, yyyy or label – TEST FLAG** – If the flag specified (z) is set (raised), the program pointer is set to the specified location (yyyy) or label. If the flag specified (z) is in the reset (lowered) condition, program execution continues with the instruction immediately following the specified location (yyyy) or label.

2nd T FLG **z, yyyy or label – TEST FOR RESET FLAG** – Reverses the sense of the test flag instruction. The transfer occurs if the flag specified (z) is reset.

What is a flag, and what is meant by flagging? A flag is a two-position signal, set or reset, which may be controlled and tested independently of a displayed number or the content of any memory register. When one or more flags are used to control the execution of a program, the siguation is referred to as *flagging*. In the course of executing a program if a flag is set (raised), then the calculator performs one set of instructions. If the flag is reset (lowered), a different set of instructions is performed depending on how the test flag (T FLG) instruction is used.

A test flag instruction does not reset the flag or affect the contents of the display or memory registers. Three of the ten flags have control features enabling program control of the **TRACE**, **DEG MODE**, and **LIMITED PRECISION** key functions.

Flag 7 controls whether radians or degrees are used by the trigonometric or conversion functions. If flag 7 is reset, then calculations are done with radians. If flag 7 is set, then degrees are used. The **DEG MODE** key effectively sets and resets flag 7.

Flag 8 controls the limited precision function. If it is reset, the full precision of each result is retained. If it is set, only the displayed value is retained. The **LIMITED PRECISION** key effectively sets and resets flag 8.

Flag 9 controls the trace mode. If it is reset, results are printed only by the **PRINT** key. If it is set, then results are printed after each new function or operation. The **TRACE** key effectively sets and resets flag 9. Do not use flags 7, 8, 9 unless program control of these functions is desired.

There are ten flags numbered 0 through 9 available for your use in programs. These flags are initially reset (lowered) and can be used to track which path a program took to completion or to control program options from the keyboard prior to running a program. They can be used to sort and discriminate against certain results.

Example: Design a program sequence to sum all incoming numbers, but print only the positive ones and display the sum after each keyboard entry.

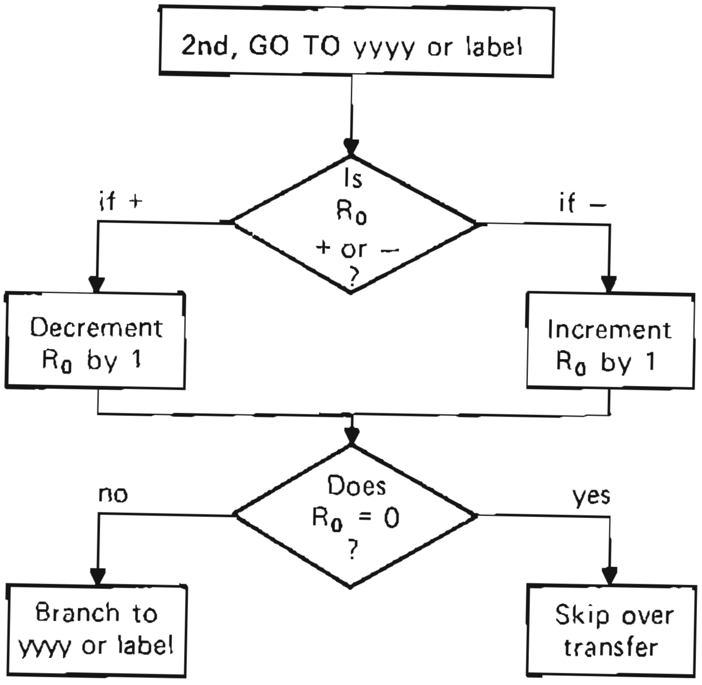
Location	Key Sequence	Comments
0018	LABEL	
0019	e_1	
0020	2nd	
0021	S FLG	Reset flag 3
0022	3	
0023	IF POS	"Is number non-negative?"
0024	e_2	If so, go to e_2
0025	S FLG	If negative, set flag 3
0026	3	
0027	LABEL	
0028	e_2	
0029	SUM	Sum all numbers
0030	2	
0031	T FLG	"Is flag 3 set?"
0032	3	
0033	e_3	If yes (number is negative), go to e_3
0034	PRINT	If not, print number
0035	LABEL	
0036	e_3	
0037	RCL	
0038	2	
0039	RTN	Halts and displays sum

Be sure register 2 is clear before entering a new series of numbers. Entering a number and pressing e_1 sums that number into data register 2, prints the number if its positive and displays the total of all entries.

Decrement and Skip on Zero (DSZ)

2nd GO TO yyyy or label – DECREMENT AND SKIP ON ZERO – Decreases the magnitude of R_0 (contents of data register 0) by 1 and processing transfers to location yyyy or to a specified label when R_0 is not 0. The transfer is skipped when the contents of register 0 reach 0.

This powerful programming instruction is an effective counter as well as a test instruction. If you need to iterate (repeat) a sequence x number of times, just store x in a data register 0 and program a DSZ test into the iterative sequence. After x iterations the looping ceases and the program continues. The DSZ instruction operations as follows:



Instruction encountered in program

Is R + or –?

Reduce magnitude of R by one

Does R = 0?

If R is not 0, transfer to label or yyyy

If R is 0, skip the transfer and continue processing.

Decrement and Skip on Zero (DSZ) Instruction

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, x, through a sequence, enter x into register 0 initially and perform DSZ last or perform DSZ first, but initially store x + 1 in register 0.

Example: Count from 10 to 1 in the display

Location	Key Sequence	Comments
0000	1	
0001	0	
0002	STORE	Store 10 in register 0
0003	LABEL	
0004	e_1	
0005	RECALL	Recall R_0
0006	PAUSE	
0007	2nd	
0008	GO TO	Go to e_1 if $R_0 = 0$
0009	e_1	
0010	HALT	Skip e_1 transfer if $R_0 = 0$

Exit the learn mode and press QUE to start the program. Try this example again to see what happens when you start with −10.

Short-Form Addressing with Transfer Operations

When using short-form addressing with transfer instructions, it is important to remember that the short form address must be followed by a nonnumeric instruction. For example, if a numeric entry must follow an IF POS instruction, short-form addressing requires a space-holder instruction.

Location	Key Code
0040	=
0041	IF+
0042	5
0043	RUN
0044	3
0045	9

CORRECT

Location	Key Code
0040	=
0041	IF+
0042	5
0043	3
0044	9
0045	6

INCORRECT

The correc
test result
holder) is
lator check
will occur
expanded
the IF+ in

Except fo
the progra
necessity.

INDIREC

This secti
simple wa
two types
addressin
expands
technique

Indirect

To unde
STORE,
hand, all
STORE,
Consequ

would r

The correct example above illustrates short-form addressing with the transfer instruction IF+. When the test result is true, the program is transferred to location 0005. When false, the RUN instruction (space-holder) is executed and the program continues. The incorrect example is erroneous because the calculator checks for the presence of an address or label following a transfer instruction, a flashing display will occur because transfer is made to location 53, 539 or 5396 depending on the partition. In the expanded calculator, the incorrect instruction sequence appears to have the address 5396 following the IF+ instruction which may also be invalid

Except for SBR, the space-holder instruction may be any nonnumeric function key that is required in the program. When short form addressing is used with the subroutine instruction, the space-holder is a necessity.

INDIRECT ADDRESSING

This section explains SR-60A indirect addressing. Several examples are given which will demonstrate some simple ways in which indirect addressing can be used to extend the power of your calculator. There are two types of indirect addressing available for your use. The first type is called *indirect data-register addressing*. The second type is called *indirect program addressing.* Although indirect addressing greatly expands programming capabilities there is some time deterioration in program running time when this technique is used.

Indirect Data Register Addressing

To understand indirect data-register addressing, recall that a direct addressing instruction such as 5, **STORE, 10** means to store 5 directly into register 10, as shown below. Indirect addressing, on the other hand, allows you to store the address of another data register for future use. For example, 5, **IND, STORE, 10** means to store 5, not in register 10, *but in the register whose address is found in register 10.* Consequently, if 15 had been previously stored in data register 10, then the instruction

5 K IND STORE **10**

would result in 5 being stored in register 15.

DIRECT

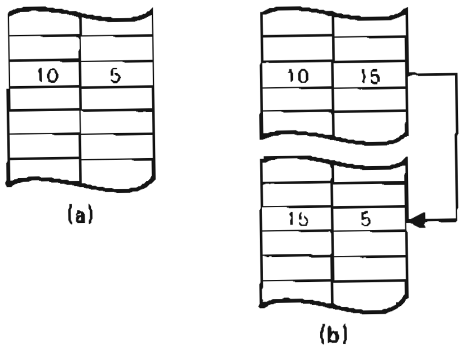
5 STORE 10

INDIRECT

5 X IND STORE 10

Data Register
No. Content

Data Register
No. Content



Direct Versus Indirect Addressing

This means that with a single indirect instruction, you can control several data registers rather than just the one you have using direct addressing.

Example: Construct a simple program to store five consecutive entries in data registers 25 through 29 using only the **e₁** key after using **QUE** to start the program.

Program Listing

Location	Key Code
0000	STO
0001	3
0002	9
0003	RTN
0004	LBL
0005	e1
0006	IND
0007	STO
0008	3

Location	Key Code
0009	9
0010	RCL
0011	3
0012	9
0013	+
0014	1
0015	=
0016	RST

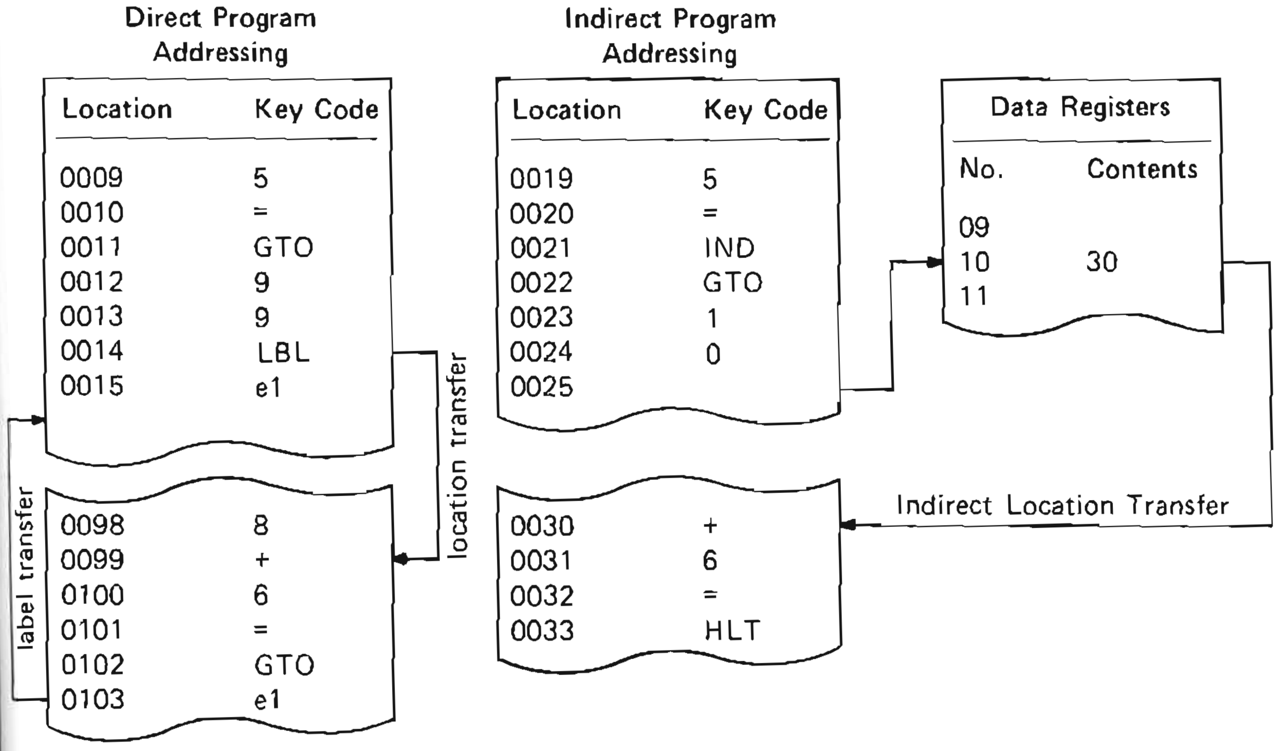
User Instructions

Enter	Press	Display
25	T QUE	25.
Entry 1	A e_1	26.
Entry 2	A e_1	27.
Entry 3	A e_1	28.
Entry 4	A e_1	29.
Entry 5	A e_1	30.

In this example, the first data register number is keyed in prior to pressing QUE. This sets the starting point which could be any register desired. The number displayed after each entry is the data register in which the next entry will be stored. Notice that you could indirectly sum, subtract, multiply, or divide with the same program by changing the instruction at address 0007.

Indirect Program Addressing

The second type of indirect addressing possible on your SR-60A is indirect program addressing. This type of indirect addressing is used to extend the usefulness of transfer instructions. Recall that there are two ways to specify a transfer address: the actual numerical location (yyyy) in program memory (absolute address) or by label. Indirect program addressing permits only one method which is to specify the data register in which the desired address is stored as shown.



Direct Program Addressing

Location	Key Code
0009	5
0010	=
0011	GTO
0012	9
0013	9
0014	LBL
0015	e1
0098	8
0099	+
0100	6
0101	=
0102	GTO
0103	e1

Indirect Program Addressing

Location	Key Code
0019	5
0020	=
0021	IND
0022	GTO
0023	1
0024	0
0025	
0030	+
0031	6
0032	=
0033	HLT

Data Registers

No.	Contents
09	
10	30
11	

Direct Versus Indirect Program Addressing

The data registers are used to store absolute addresses of program locations in addition to their regular duties. Thus, the address which is explicitly a part of a transfer statement in indirect addressing refers to a data register address. This means that labels cannot be used in the same way they are used in direct addressing.

Indirect Unconditional Transfers

IND, GO TO, 0nnn

IND, SUBR, 0nnn

Notice that the same number of address digits are required for indirect addressing as for direct or absolute addressing. Here nnn is the data register address where the transfer address is stored.

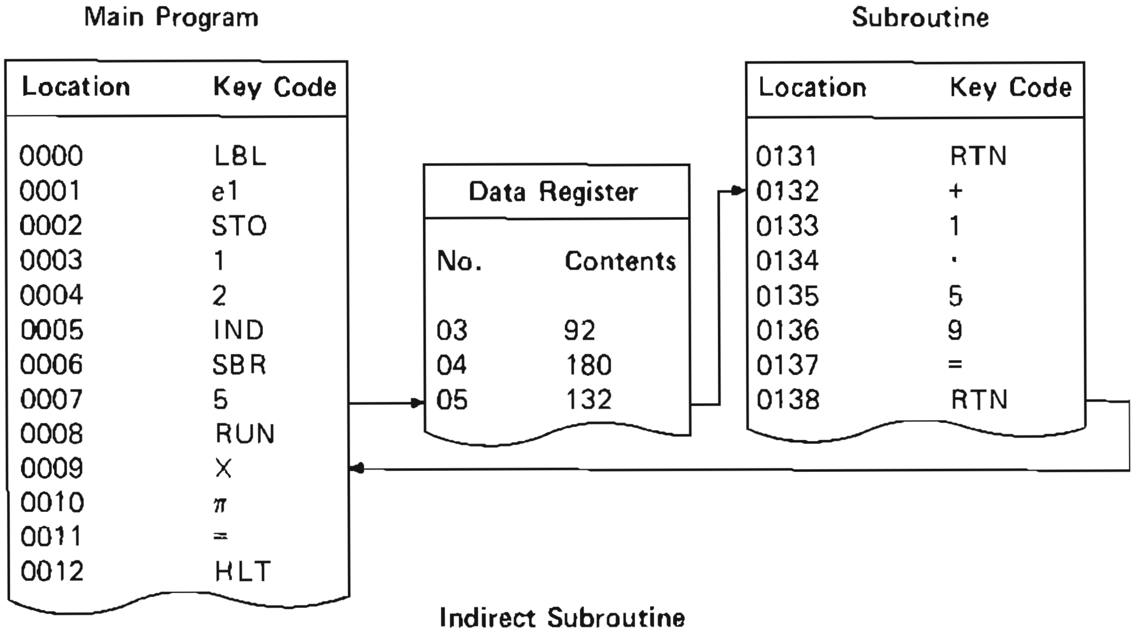
Indirect Conditional Transfers

IND, IFF ERR, 0nnn
IND, IF POS, 0nnn
IND, IF ZRO, 0nnn
IND, T FLG z, 0nnn

IND, 2nd, IF ERR, 0nnn
IND, 2nd, IF POS, 0nnn
IND, 2nd, IF ZRO, 0nnn
IND, 2nd, T FLG z, 0nnn

Except for **SUBR**, the short form data register address for nnn may be used if a non-numerical entry follows the complete instruction sequence.

The following illustrates a partial program listing that uses the indirect addressing of a subroutine.



Main Program

Location	Key Code
0000	LBL
0001	e1
0002	STO
0003	1
0004	2
0005	IND
0006	SBR
0007	5
0008	RUN
0009	X
0010	π
0011	=
0012	HLT

Data Register

No.	Contents
03	92
04	180
05	132

Subroutine

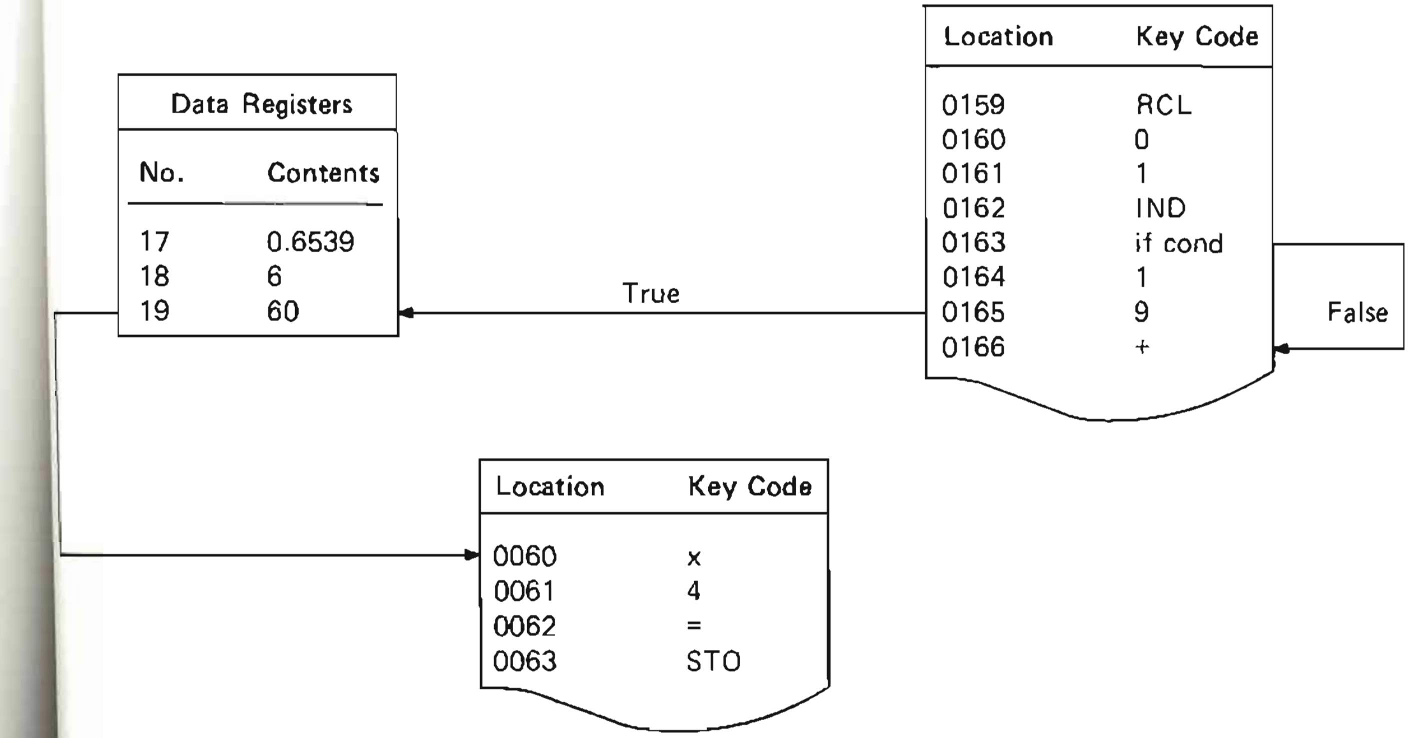
Location	Key Code
0131	RTN
0132	+
0133	1
0134	·
0135	5
0136	9
0137	=
0138	RTN

Indirect Subroutine

Note that short form indirect addressing is possible with the subroutine instruction, providing a space-holder instruction follows the short form address. The **RUN** instruction in location 0008 serves this purpose and the return address in the subroutine return register is 0009. The space-holder instruction is necessary with any indirect subroutine instruction which uses short-form addressing.

Conditional transfers, including the test-flag instruction, may also be used with indirect addressing. Below is a partial program listing for the operation of indirect conditional transfers.

The instruction at location 0163 could be any one of the conditional transfer instructions. Remember that the T **FLG** (TFS) instruction requires the flag number (z) before the indirect address (nnnn).



Data Registers

No.	Contents
17	0.6539
18	6
19	60

Location	Key Code
0159	RCL
0160	0
0161	1
0162	IND
0163	if cond
0164	1
0165	9
0166	+

Location	Key Code
0060	x
0061	4
0062	=
0063	STO

Indirect Conditional Transfer

Data register 19 is indirectly called without a zero prefix because a nonnumeric instruction is at location 0166.

IMPORTANT: The number stored in a data register for indirect addressing is limited by your partition. Indirectly addressing a data register with a stored number that is larger than 1 X 10^5 *will cause the display to blank out for a time period that increases with the magnitude of the number. The* **CLEAR ALL** *key may be used to regain keyboard control, but this destroys the program and the contents of the data registers. So, it is best to avoid such a situation.*

Details IV

PROGRAM PROMPTING

The programs in the previous section illustrate the generation and use of non-prompting program techniques. While most are quite simple to run, notice that you needed to remember what key to press. Also, there is seldom a printout generated by the program for a permanent record of calculation results. Consider the difficulty in remembering key entries, etc., for a program which requires many entries in a specific order. A detailed written procedure would be needed until memorized. Now consider how simple it would be to run a program where the calculator is able to display messages and questions pertaining to the exact entry it needs to continue the program. And in addition, a detailed printout identifies entries and results by name. The prompting functions of the SR-60A permit this type of program operation. A prompting operation frees you from having to memorize lengthy procedures because the SR-60A can display messages or questions in plain English (or any other language using the same alphabet) and effectively lead you step-by-step through complex calculations. This section provides a description of the calculator functions directly related to this prompting.

ALPHANUMERIC OPERATIONS

Prompting messages are initiated and terminated by the **ALPHA** key. The key sequence to generate a message is the same whether used in a program or manually on the keyboard.

ALPHA **ALPHANUMERIC** — Used to start and end an alphanumeric field. The first and succeeding alternate strokes of this key cause key entries to be placed in the alpha display register in a left-to-right order. The second and alternate strokes of this key terminate the alphanumeric field and restore normal keyboard operation. **ALPHA** key entries must always be used in pairs for proper operation. A pair is as follows:

Normal Operation ALPHA Alphanumeric Entries ALPHA Normal Operation

There are several important aspects related to alphanumeric entries. Keep the following points in mind when making alphanumeric entries:

1. A maximum of 20 characters per message (including spaces) can be entered into an alphanumeric field. Any entries resulting in more than 20 characters per message are lost.

2. Attempt to enter more than 23 characters will automatically terminate the alphanumeric field, cause a flashing display, and execute the normal function of the key entry causing the flashing display.

3. The **CLEAR ALL** or **CLEAR** key may not be used within an alphanumeric field to produce alphanumeric characters. The **CLEAR** key will terminate an alphanumeric field. However, it produces an unmatched alpha instruction in a program and will cause improper operation. The **CLEAR ALL** key clears everything, always, in or out of alpha mode.

4. An erroneous entry while making alphanumeric entries can be corrected by terminating the alphanumeric field and starting over or by using **2nd, STORE** (see *Editing Alphanumerics* later in this section).

5. Program steps that make up an alphanumeric field in a program may be edited (note item 4) like any otherpart of the program unless it is necessary to enter the letters **U, V, W, X,** or **Z** (control keys). The **2nd, STORE** procedure must be used here or reenter the entire alpha field.

Each key entry in an alphanumeric field produces one, two, or three characters representing the key. The single-character alphabet entries are made by the programming keys on the right side of the calculator. The upper half of each key is marked with the letter or symbol produced in an alphanumeric field. Numbers and operations are produced by the number and operation keys and also produce single-digit characters. The figure below illustrates the alphanumeric representation of the other keys when used in an alphanumeric field. The **DEG MODE, PRINT, TRACE, PAPER ADV** and **LIMITED PRECISION** keys produce special symbols and punctuation marks that are not represented on any of the keys. Notice that the identification for each key takes one or three character locations in the display. A space will always appear following a key that is represented by two characters.

Key	Display	Key	Display	Key	Display	Key	Display
DEG MODE	°	e_{12} COS	COS	10^x	10^x	%	%
PRINT	/	e_{13} TAN	TAN	1/x	1/X	YES	YES
TRACE	*	e_{14} π	π	√x	√‾ X_†	NO	NO_†
PAPER ADV	, (Comma)	e_{15} P/R	P/R	CLEAR MEM	CMS	NOT APPLY	N/A
LIMITED PRECISION	' (Apostrophe)	2nd	II	EXCH	XM_†	NOT KNOWN	UNK
e_6 D/R	D/R	Ln x	LNX	PROD	Π	ENTER	ENT
e_7 ARC	ARC	LOG	LOG	Int x	INT	CE	CE_†
e_8 HYP	HYP	x!	X!_†	Δ%	Δ%_†	FIX	FIX
e_9 DMS	DMS	x^2	X^2_†	STORE	STO	EE	EE_†
e_{10} x⇄R	X ⇄ R	x⇄K	X ⇄ K	RECALL	RCL	y^x	Y^x_†
e_{11} SIN	SIN	e^x	e^x_†	SUM	Σ	$^x\sqrt{y}$	$^x\sqrt{\ }$Y

†The open underline indicates an automatic character space.

Alphanumeric Key Identification

These keys can be valuable for abbreviated alpha entry.

Example: Key in the alphanumeric message, "STORE AN ENTRY".

Key Sequence

ALPHA ALPHA | STORE | R SUBR | E e_0 | SPACE RUN | A e_1 | N IF POS | SPACE RUN | ENTER | R SUBR | Y RESET | ALPHA ALPHA

The alphanumeric message remains in the display after the second (closing) **ALPHA** key is entered until another key is pressed and the display reverts to the numeric display register. The only exception is the **PAPER ADV** key, which does not affect the display. When a mathematical function key is pressed while a message is displayed, the function is performed on the contents of the numeric display register and the result is then displayed.

When executed as part of a program, the PAUSE, HALT and RETURN instructions may be executed one time without losing the displayed alphanumeric message. The one-time execution limit has only one possible exception and that is the QUEUE instruction as described in following paragraphs. The result is that if two PAUSE instructions are executed following an alphanumeric message, the first pause will display the message and the second will display the numeric-display register contents. The same sequence using the PRINT instruction will behave the same way and has some advantages.

Example: Construct a simple program to solve the equation X + 5% with the input value of X, the equation and result to be printed.

Location	Key Code
0000	LBL
0001	e1
0002	ALF
0003	X
0004	=
0005	ALF
0006	PRT
0007	PRT
0008	+
0009	5
0010	%
0011	=

Location	Key Code
0012	ALF
0013	X
0014	+
0015	5
0016	%
0017	=
0018	ALF
0019	PRT
0020	PRT
0021	RTN

Press

100 A e_1

Print

X = 100.
X + 5% = 105.

In this example, the double PRINT instruction is used in two places. The first alphanumeric message (X =) is printed by location 0006 and the entered value of X is printed by location 0007. Then the actual result is calculated by locations 0008 through 0011 followed by the second message and two PRINT instructions. The program would have operated just as well if the calculation in locations 0008 through 0011 were moved to a position between the two print instructions at 0019 and 0020. Both ways have some advantages. The double print instruction prevents attracting user attention to the printer when the numerical result may not be printed until some time later. On the other hand, placing the alphanumeric field and print instruction before a long calculation will cause the message to flicker in the display while the calculation is in progress. The choice really depends on personal preference.

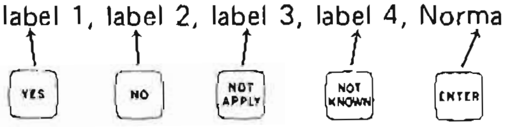
Another thing to note from the previous example is that a number in the display (display register) prior to generating an alphanumeric message is easily recovered following the message.

PROMPTING CONTROL (QUE) AND RESPONSES

In addition to the keys related to generating alphanumeric messages, there are six keys which directly control the calculator prompting operations: **QUE, YES, NO, NOT APPLY, NOT KNOWN** and **ENTER.** Basically the QUE key is the control function and the others are designated as response keys.

T QUE QUEUE – When executed in a program, the program stops with the contents of the alpha display register in the display. The QUEUE instruction must be followed by four labels which correlate with program segments to be executed depending upon which response key is pressed.

QUE, label 1, label 2, label 3, label 4, Normal instructions



When **YES** is pressed, program operation transfers to label 1, **NO** to label 2, etc. The **ENTER** key simply causes the program to resume with the fifth location after the **QUE.**

The transfer control of the QUEUE instruction can be limited to one label if response is required of only one of the response keys.

Response Key, **QUE,** Label, Normal instructions

In this case, a response key (**YES, NO, NOT APPLY, NOT KNOWN** or **ENTER**) stored immediately before the QUEUE instruction will enable a transfer to the label when that particular response key is pressed. All other response keys cause the program to continue with the next instruction following the QUEUE and label instructions.

YES NO NOT APPLY NOT KNOWN ENTER **PROMPTING RESPONSES** – These keys are used to manually respond to a message or question displayed following execution of a QUEUE instruction. Refer to *Guidelines for Prompting* in this section for a description of the ways the response may be used.

The following diagram illustrates the two versions of the QUEUE instruction.

```
Response Key                Key Code              Comments

                            .
                            .
                            .
                            ALF
                            .
                            .
                            Alphanumeric
                            Message
                            .
                            ALF
                            QUE                   Program stops and
                                                  displays message
YES . . . . . . . . . . ──→ e4.  ──→ . . . . . . . Transfer to label e4
NO  . . . . . . . . . . ──→ D/R  ──→ . . . . . . . Transfer to label D/R
NOT APPLY . . . . . . . ──→ D/R  ──→ . . . . . . . Transfer to label D/R
NOT KNOWN . . . . . . . ──→ 10^x ──→ . . . . . . . Transfer to label 10^x
ENTER . . . . . . . . . ──┐ +                     Execute  +  and continue
                          ↓ 2                     program
                            =
                            .
                            .
                            ALF
                            .
                            .
                            Alphanumeric
                            Message
                            .
                            .
                            .
                            ALF
                            YES                   Presets QUE to transfer
                                                  only on a YES response
                            QUE                   Program stops and
                                                  displays message
YES . . . . . . . . . . ──→ e4.  ──→ . . . . . . . Transfer to label e4
NO  . . . . . . . . . ──┐── +                     All other responses execute +
NOT APPLY . . . . . . ──┤ ↓ 2                     and continue program
NOT KNOWN . . . . . . ──┤   =
ENTER . . . . . . . . ──┘
```

QUEUE Instruction Format

The first form with four labels is the most versatile, but the second form can save program space when only a yes or no response is required. Note that while the alphanumeric message is shown in the program immediately before the QUEUE instruction, the QUEUE instruction will cause the last entered alphanumeric message in the alpha display register to appear without regard to when it was entered. For example, you can enter a message from the keyboard, then run a program which has a QUEUE instruction without a stored alpha field and the keyboard message will appear again.

Example: What is the procedure which allows an alphanumeric message entered from the keyboard to be displayed and printed more than one time without reentering the message?

Key Code

```
LBL
e1
QUE
```

When stored in program memory, this simple three-step program will cause the last stored alphanumeric message to reappear when e_1 is pressed. This operation does not allow using the response keys unless more instructions are added as shown in the diagram. Also note that if the alpha-display register is empty (after using **CLEAR ALL** or pressing **ALPHA** twice in succession), the display will be blank if e_1 is pressed to execute the QUEUE instruction. Simply press another key such as CE or = to restore the numeric display register contents to the display.

GUIDELINES FOR PROMPTING

There are many ways to set up a prompting program. The versatility of the QUEUE instruction and the alphanumeric message make it necessary to establish general guidelines for messages used in programs. The following guidelines have been developed for standardization of the prerecorded library programs in the Basic Library and the optional libraries.

Program Title

When required, program titles should print automatically when the card is loaded. The title should be preceded by a paper advance and can be set off from other messages by beginning with two asterisks.

Examples: **COMPOUND INTEREST
**STATISTICS

Questions

Questions of the yes/no variety appear as written in general usage. A question mark terminates the question word or phrase and the response should be with the **YES** or **NO** key. Questions are not usually printed.

Examples: DELETIONS?
                 MARRIED?

Data Entry

Data entry is prompted using two methods. The first method is used whenever program space permits separate messages for prompting and printout. Beginning a message with the word ENTER implies the program needs additional data and the **ENTER** key is to be pressed after keying in the data. In some cases the response could be **NOT KNOWN** or **NOT APPLY** which means the value called by the message is to be solved for or ignored by the program.

Examples: ENTER INTEREST (%)
                 ENTER VALUE OF X

To key in the word "ENTER" only takes 3 keystrokes, **ENTER, E, R.**

A second method to ask for data is designed to be used as a printout and a message to enter data. An equals sign is used with a message to indicate data entry is expected. The same message can be printed to identify the entered data.

Examples: INTEREST (%) =
                 VALUE OF X =

Results

Results are printed with an appropriate alphanumeric name.

Examples: FUTURE VALUE
                 NET PAY

It is a good practice to indicate when a program completes all calculations. The common identifier for the end of a program is to print three asterisks after the final result.

BASIC PROMPTING EXAMPLES

Example 1: Develop a prompting program to calculate the area of a rectangle. The length of two sides of the rectangle are to be asked for by the program with the sides and the resulting area printed.

Equation: Length X Width = Area

Prompting Messages:
LENGTH = (Print and display)
WIDTH = (Print and display)
AREA (Print for result)

Conditions: Program to be initiated by using the QUE key.

Location	Key Code
0000	PA
0001	ALF
0002	L
0003	E
0004	N
0005	G
0006	T
0007	H
0008	=
0009	ALF
0010	PRT
0011	LBL
0012	LNX
0013	QUE
0014	LNX
0015	LNX
0016	LNX
0017	LNX
0018	STO
0019	PRT

Location	Key Code
0020	ALF
0021	W
0022	I
0023	D
0024	T
0025	H
0026	=
0027	ALF
0028	PRT
0029	LBL
0030	LOG
0031	QUE
0032	LOG
0033	LOG
0034	LOG
0035	LOG
0036	Π
0037	PRT

Location	Key Code
0038	ALF
0039	A
0040	R
0041	E
0042	A
0043	ALF
0044	PA
0045	PRT
0046	RCL
0047	PRT
0048	ALF
0049	*
0050	*
0051	*
0052	ALF
0053	PRT
0054	RTN

To enable the program to start by using the **QUE** key, instructions start at location 0000. The first instruction is paper advance (**PA**). Paper advance at the start of a program takes slack out of the printer drive system which may result from previously tearing the paper off. If the drive system has slack, the characters in the first line printed may appear with flat tops or even as dashes.

Locations 0001 through 0010 enter and print the first prompting message "LENGTH=". The QUEUE instruction at 0013 halts the program with the message displayed. In this case, the only response desired is a numerical key entry followed by the **ENTER** key. Since there must be labels in the first four locations following the **QUE** key, the ideal situation is to have labels such that if any response key other than **ENTER** is pressed, the program is not affected. Preceding the **QUE** with **LABEL**, **Lnx** and following QUE with **Lnx** if the four label positions permits such operation. If **YES** is pressed, the program simply executes the QUEUE instruction again and waits for another response. When the value for the length is keyed in and **ENTER** is pressed, it is stored in R_{00} at 0018 and printed at 0019.

In similar manner, the width is asked for by location 0020 through 0035. Label LOG is used with the QUEUE instruction to control improper responses. As with any transfer instruction, if the QUEUE instruction is followed by a label that is not identified in program memory with the label key, a question mark will flash in the display.

The width value is multiplied times the height in R_{00} by the product instruction at 0036. The width is then printed at 0037. Locations 0038 through 0045 enter and print the message "AREA" to identify the result. The paper advance instruction was used to set the result apart from the entered values. The area result is recalled and printed at locations 0046 and 0047. The remaining instructions cause three asterisks to be printed to indicate end of program.

One other item to point out about the preceding program is that entries and computation with R_{00} has several advantages over using normal arithmetic keys. If a multiply instruction were placed at 0018, the product instruction at 0036 deleted, and an equals placed at 0046, the program would still operate. However, if an interim calculation were performed following the second QUEUE instruction, or even the **CLEAR** key used, the result would not be correct. The above program permits any interim operation except use of R_{00} or **CLEAR MEM**. This also means the program can be used to calculate area in the middle of a problem without affecting pending operations.

Use the prompting program to calculate the area of a window that is 29 by 42 inches.

Press	Display	Print
T QUE	LENGTH =	LENGTH =
29 ENTER		29.
	WIDTH =	WIDTH =
42 ENTER		42.
		AREA
		1218.
	1218.	***

Now solve the same problem with the calculator trace operation selected. Notice that the ALPHA instructions that ends an alphanumeric field automatically prints the message such that the following PRINT instruction causes the contents of the numeric display register to be printed. Also note the different way of starting the program for trace operation because the **QUE** key will automatically turn trace off because it resets the flags.

Press	Display	Print
CLEAR CLEAR MEM Y RESET	0	
TRACE (light on)	0	0
SPACE RUN		
		LENGTH =
		0. PRT
		LBL
	LENGTH =	LENGTH =
29 ENTER		ENT
		29. STO
		0
		29.
		29. PRT
		WIDTH =
		29. PRT
		LBL
	WIDTH =	WIDTH =
42 ENTER		ENT
		42. Π
		0
		42.
		42. PRT
		AREA
		42. PRT
		RCL
		0
		1218.
		1218. PRT
		***
		1218. PRT
	1218.	1218.
TRACE (light off)	1218.	1218. TRC

Details

To aid your understanding of the program in this example, develop your own version of a flow diagram for the program before continuing to the next example.

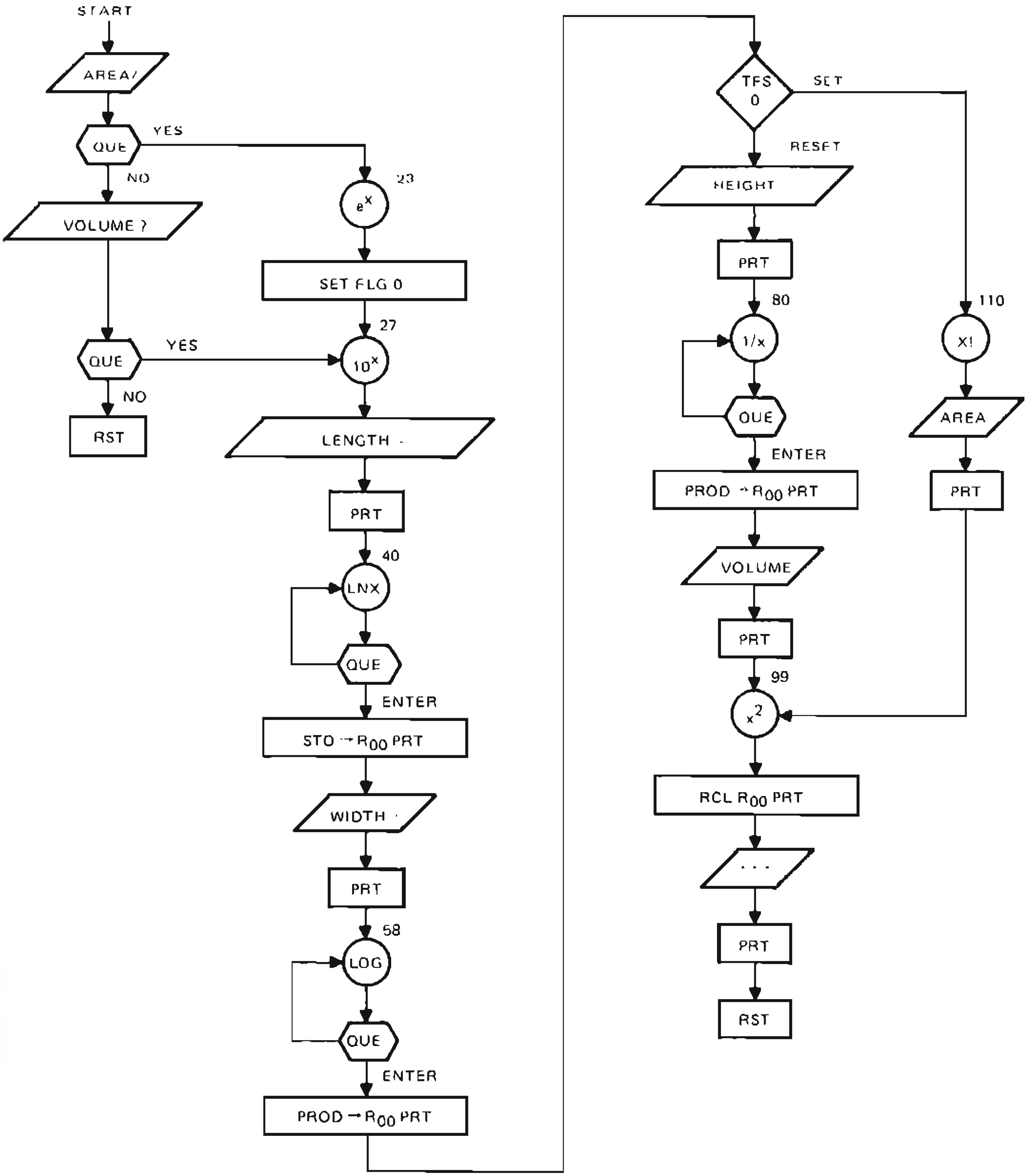
Example: Expand the program in the previous example to calculate area of a rectangle or the volume of a box.

Equations: Length X Width = Area
Length X Width X Height = Volume

Prompting Messages:	AREA?	(Display)
	VOLUME?	(Display)
	LENGTH =	(Print and display)
	WIDTH =	(Print and display)
	HEIGHT =	(Print and display)
	AREA	(Print for result)
	VOLUME	(Print for result)

Conditions: Automatic program restart

A detailed flow diagram of the expanded program is shown below. Notice the additional types of symbols compared to previous flow diagrams. As stated before, there are no set rules for flow diagrams, the symbols in this manual were arbitrarily chosen to simplify interpretation. The new symbols are parallelograms for prompting messages and hexagons for QUEUE instructions.



Detailed Flow Diagram

Details

The first part of the program uses the second version of the QUEUE instruction where, in this case, a transfer will occur only when the YES key is pressed. The use of flag 0 permits sharing parts of the two calculations that are the same thus saving space. The reset instruction at the end of the program is very important. It causes the program to automatically restart and it also resets flag 0 so the program will operate properly on subsequent runs. To convert the program to function as a subroutine, a return instruction would be required in place of reset and a reset flag 0 instruction would also be required before the program restarts.

After entering the key code sequence for the program into the calculator, find the area of a door that is 3.25 feet by 6.5 feet. Then find the volume of a carton that is 4.25 X 3 X 1.5 feet.

Press	Display	Print
T QUE	AREA?	
YES	LENGTH =	LENGTH =
6.5 ENTER		6.5
	WIDTH =	WIDTH =
3.25 ENTER		3.25
		AREA
		21.125
	AREA?	***
NO	VOLUME?	
YES	LENGTH =	LENGTH =
4.5 ENTER		4.5
	WIDTH =	WIDTH =
3 ENTER		3.
	HEIGHT =	HEIGHT =
1.5 ENTER		1.5
		VOLUME
		20.25
	AREA?	***

Location	Key Code	Location	Key Code	Location	Key Code
0000	ALF	0041	LNX	0082	QUE
0001	A	0042	QUE	0083	1/X
0002	R	0043	LNX	0084	1/X
0003	E	0044	LNX	0085	1/X
0004	A	0045	LNX	0086	1/X
0005	?	0046	LNX	0087	Π
0006	ALF	0047	STO	0088	PRT
0007	YES	0048	PRT	0089	PA
0008	QUE	0049	ALF	0090	ALF
0009	e^x	0050	W	0091	V
0010	ALF	0051	I	0092	O
0011	V	0052	D	0093	L
0012	O	0053	T	0094	U
0013	L	0054	H	0095	M
0014	U	0055	=	0096	E
0015	M	0056	ALF	0097	ALF
0016	E	0057	PRT	0098	PRT
0017	?	0058	LBL	0099	LBL
0018	ALF	0059	LOG	0100	X^2
0019	YES	0060	QUE	0101	RCL
0020	QUE	0061	LOG	0102	PRT
0021	10^x	0062	LOG	0103	ALF
0022	RST	0063	LOG	0104	*
0023	LBL	0064	LOG	0105	*
0024	e^x	0065	Π	0106	*
0025	SF	0066	PRT	0107	ALF
0026	O	0067	TFS	0108	PRT
0027	LBL	0068	O	0109	RST
0028	10^x	0069	X!	0110	LBL
0029	PA	0070	ALF	0111	X!
0030	ALF	0071	H	0112	PA
0031	L	0072	E	0113	ALF
0032	E	0073	I	0114	A
0033	N	0074	G	0115	R
0034	G	0075	H	0116	E
0035	T	0076	T	0117	A
0036	H	0077	=	0118	ALF
0037	=	0078	ALF	0119	PRT
0038	ALF	0079	PRT	0120	GTO
0039	PRT	0080	LBL	0121	X^2
0040	LBL	0081	1/X		

Key Code Sequence

STORING ALPHANUMERICS

It is possible to save alphanumeric information by storing it in data registers. You can store specific information for later use either from the keyboard or from a program. Simply key in an alphanumeric message as previously discussed, then take the characters, 5 at a time, and store them into whichever data registers you desire. You specify which set of 5 you are going to store by the designation below.

1 2 3 4 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	Character Positions
01	02	03	04	Sets of 5 characters

Display and Paper Tape

To retrieve a set of 5 characters, press **2nd EXCH** then enter the set number (01, 02, 03, 04). This exchange process is a memory operation so the set number must agree with the partition's restriction on the number of digits in the address. Alternatively, short-form addressing can be used.

Example: Store the name "JIM JONES" in data registers 11 and 12.

Press	Display	Comments
ALPHA JIM JONES ALPHA	JIM JONES	Load alpha register
2nd EXCH 01	ONES	"JIM J" taken to numeric register
STORE 11	2524280025.	Character codes* for "JIM J" stored in R_{11}
CLEAR 2nd EXCH 02	(blank)	"ONES" taken to numeric register leaving alpha register blank
STORE 12	3029203400.	Character codes for "ONES" stored in R_{12} "JIM JONES" is now stored in R_{11} and R_{12}. To reconstitute this message whenever desired, follow the remainder of the example.

*These character codes are discussed a few pages later.

Press	Display	Comments
RECALL 11	2524280025.	
2nd EXCH 01	JIM J	
RECALL 12	3029203400.	
2nd EXCH 02	JIM JONES	
PRINT	0.	Display numeric register

Notice that when the contents of the alpha display register are printed, the cleared contents of the numeric display register is displayed. The contents of the alpha register, though, remain in that register until deliberately altered or erased and can be retrieved and displayed at any time by pressing **2nd EXCH 00**. The alpha register can be cleared by pressing **ALPHA ALPHA**.

A program can just as easily use alphanumeric information both internally and from the outside.

Example: Design a program to calculate and label the final cost of items that have a 5% sales tax. The program expects to have the cost of the item in the display when QUE is pressed to start the program. The program pauses and asks for the item name or number (up to 10 characters allowed) to be entered alphanumerically.

Location and Key Code	Key Sequence	Comments
0000 STO	STORE	
0001 LBL	P LABEL	
0002 +	+	
0003 ALF	ALPHA ALPHA	
0004 I	I $ FLG	
0005 T	T QUE	
0006 E	E e_5	
0007 M	M IF ERR	
0008 ALF	ALPHA ALPHA	Asks for item name
0009 QUE	T QUE	
0010 +	+	
0011 +	+	
0012 +	+	
0013 +	+	
0014 II	2nd	
0015 XM	EXCH	Places first 5 characters in data register 1
0016 1	1	
0017 STO	STORE	
0018 1	1	
0019 II	2nd	

Location and Key Code	Key Sequence	Comments
0020 XM	EXCH	Places next 5 characters in data register 2
0021 2	2	
0022 STO	STORE	
0023 2	2	
0024 RCL	RECALL	
0025 FIX	FIX	
0026 2	2	
0027 +	+	
0028 5	5	
0029 %	%	
0030 =	=	
0031 ALF	ALPHA ALPHA	
0032 T	T QUE	
0033 O	O IF ZRO	
0034 T	T QUE	
0035 A	A e_1	
0036 L	L LIST	
0037 +	+	
0038 T	T QUE	
0039 A	A e_1	
0040 X	X DLETE	

Location and Key Code	Key Sequence	Comments	Location and Key Code	Key Sequence	Comments
0041 ALF	ALPHA		0051 RCL	RECALL	
0042 PRT	PRINT	Prints alpha	0052 2	2	Places next 5 characters of item name in 4th quarter of display
0043 PRT	PRINT	Prints results	0053 II	2nd	
0044 ALF	ALPHA	Clears alpha register	0054 XM	EXCH	
0045 ALF	ALPHA		0055 4	4	
0046 RCL	RECALL		0056 PRT	PRINT	Print item name
0047 1	1	Places first 5 characters of item name in 3rd quarter of display	0057 PA	PAPER ADV	
0048 II	2nd		0058 PA	PAPER ADV	
0049 XM	EXCH		0059 RTN	$ RTN	
0050 3	3				

Calculate the total cost of a $125 size 42 medium jacket.

Press	Display	Printer
125 ▼ QUE	ITEM	
ALPHA JACKET 42M ALPHA		
ENTER		TOTAL+TAX 131.25 JACKET 42M

EDITING ALPHANUMERICS

Each alphanumeric character is represented by a two-digit code as follows:

Units digit

Tens digit	0	1	2	3	4	5	6	7	8	9
0	blank	0	1	2	3	4	5	6	7	8
1	9	•	2	x	!	△	A	B	C	D
2	E	F	G	H	I	J	K	L	M	N
3	O	P	Q	R	S	T	U	V	W	X
4	Y	Z	*	$	?	,	%	(	=	–
5	+	÷	X	π	'	)	√‾	°	Σ	Π
6	/	⇄	e	[]						

Character Codes

For instance, A is code 16 and π is code 53. Notice that there are codes available for several characters other than those obtainable from the keyboard, like √‾ code 56 and ! code 14.

The advantage of a code being assigned to each character provides for the editing of an alphanumeric field during manual operation without having to reenter the entire message if any part needs to be changed. The key sequence to accomplish this edit is to enter the code of the character you chose to place in the alpha field and press **2nd, STORE, x** where x is one of the 20 character positions available in the display and on the printer tape as shown previously.

When keying in a message from the keyboard, there is no provision to back up and correct a faulty entry. Instead of starting over, simply complete the message and press **ALPHA** to terminate the field. Now follow the above key sequence to correct the error.

Example: Suppose you have keyed in "SIMPLE PROBLEM" instead of "SAMPLE PROBLEM" and wish to correct it.

Enter: ALPHA SIMPLE PROBLEM ALPHA

To make "SIMPLE" into "SAMPLE":

Press	Display	Comments
16	16	Enter code for "A"
2nd STORE 02	SAMPLE PROBLEM	Corrected message

To check the code of a particular alpha character, press **2nd**, **RECALL**, **x**, then any nonexecutable key. For instance, to check on the code for "P" in the above example, press **2nd**, **RECALL**, **04** and the entire message is displayed. Now press any nonexecutable key like **CE** to display the numeric code for the character in position 4.

The message in the alpha register remains there until cleared or replaced. The **2nd**, **RECALL**, **x** technique can be used to bring the message to the display for printing purposes whenever desired without affecting calculations in progress.

Caution

The partition must be set so that the number of program locations available is an even multiple of 160.

Plotting With Alphanumerics

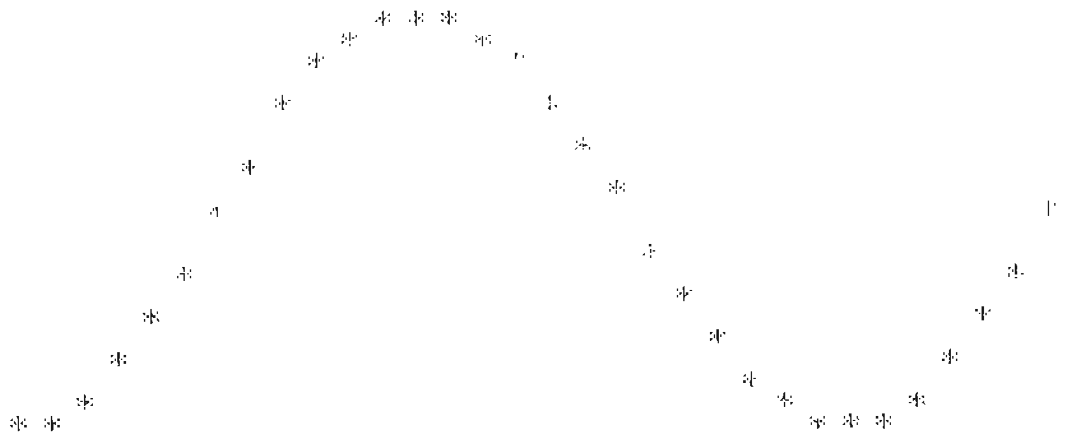
With the **2nd**, STORE key sequences discussed above, you can selectively place any of the available characters in any of the 20 positions. This technique can be used to plot data by scaling results into the 1 to 20 range of the display (paper tape), then selecting a character to print there. Actually, the **2nd**, STORE sequence can be used indirectly to accomplish a plot.

Example: Make a plot of the sine function (SIN) on the calculator. Plot a point every quarter of a radian.

Location and Key Code	Key Sequence	Comments
0000 LBL	P LABEL	
0001 e1	A e1	
0002 RCL	RECALL	Recall contents of R_0
0003 SIN	e0 SIN	Find sine of radian angle
0004 +	+	Add 1 to make all results 0 to 2
0005 1	1	
0006 =	=	
0007 X	×	
0008 9	9	
0009 •	·	Scales results up to 0 to 19.8
0010 9	9	
0011 +	+	
0012 1	1	
0013 =	=	Extends range to 1 to 20.8
0014 INT	Int x	Creates values in range 1 to 20
0015 STO	STORE	Store this result in R_6
0016 5	5	
0017 ALF	ALPHA ALPHA	Clear alpha register
0018 ALF	ALPHA ALPHA	

Location and Key Code	Key Sequence	Comments
0019 4	4	
0020 2	2	Code for a *
0021 IND	K IND	
0022 II	2nd	
0023 STO	STORE	
0024 5	5	Indirectly store * in R_5
0025 PRT	PRINT	Plot point
0026 •	•	
0027 2	2	
0028 5	5	
0029 Σ	SUM	Add .25 to R_0
0030 RST	Y RESET	Return to location 0000

Simply press **QUE** or e_1 to run the program and plot the results.



See if you can make the program plot + for positive values of sine and − for negative values.
Hint: check the sign of sine in location 0003 and set a flag to indicate negative results. The number of digits submitted for x depends on the partition as with other data memory operations from the keyboard. Short-form addressing should be used whenever possible, especially in programs, not only to save space, but to eliminate dependence on the partition. For instance, **STORE, 5, +** works for any partition, whereas **STORE, 005, +** only works when the partition provides more than 99 data registers.

Alphabetizing Program

Occasionally, it is desirable to analyze alphanumeric characters and make decisions based upon them. This is done by looking at the respective character codes.

Example: Design a program to alphabetize left justified, five character words. First, receive all the words into the program, then alphabetize and print them.

Location and Key Code	Key Sequence	Location and Key Code	Key Sequence	Location and Key Code	Key Sequence
0000 5	5	0016 STO	STORE	0032 RCL	RECALL
0001 1	1	0017 RCL	RECALL	0033 IND	K IND
0002 STO	STORE	0018 3	3	0034 RCL	RECALL
0003 1	1	0019 −	−	0035 2	2
0004 STO	STORE	0020 RCL	RECALL	0036 −	−
0005 3	3	0021 1	1	0037 IND	K IND
0006 STO	STORE	0022 =	=	0038 RCL	RECALL
0007 4	4	0023 STO	STORE	0039 1	1
0008 RTN	$ RTN	0024 RCL	RECALL	0040 =	=
0009 LBL	P LABEL	0025 1	1	0041 IF +	N IF POS
0010 e5	E e5	0026 +	+	0042 +	+
0011 1	1	0027 1	1	0043 IND	K IND
0012 II	2nd	0028 =	=	0044 RCL	RECALL
0013 Σ	SUM	0029 STO	STORE	0045 1	1
0014 3	3	0030 2	2	0046 X ⇄ R	e10 x⇄R
0015 LBL	P LABEL	0031 LBL	P LABEL	0047 IND	K IND

Location and Key Code	Key Sequence
0048 RCL	RECALL
0049 2	2
0050 IND	K IND
0051 STO	STORE
0052 1	1
0053 X ⇄ R	e_{10} X⇄R
0054 IND	K IND
0055 STO	STORE
0056 2	2
0057 GTO	$ GO TO
0058 STO	STORE
0059 LBL	P LABEL
0060 +	+
0061 1	1
0062 Σ	SUM
0063 2	2
0064 II	2nd
0065 GTO	$ GO TO
0066 RCL	RECALL
0067 1	1
0068 Σ	SUM
0069 1	1
0070 RCL	RECALL
0071 3	3
0072 −	−
0073 RCL	RECALL
0074 1	1
0075 =	=
0076 II	2nd
0077 IF ZRO	0 IF ZRO
0078 STO	STORE
0079 RCL	RECALL
0080 2	2
0081 −	−
0082 RCL	RECALL
0083 4	4
0084 =	=
0085 STO	STORE
0086 LBL	P LABEL
0087 PRT	PRINT
0088 IND	K IND
0089 RCL	RECALL
0090 4	4
0091 II	2nd
0092 XM	EXCH
0093 1	1
0094 PRT	PRINT
0095 1	1
0096 Σ	SUM
0097 4	4
0098 II	2nd
0099 GTO	$ GO TO
0100 PRT	PRINT
0101 RTN	$ RTN
0102 LBL	P LABEL
0103 e1	A e_1
0104 II	2nd
0105 XM	EXCH
0106 1	1
0107 IND	K IND
0108 STO	STORE
0109 3	3
0110 RUN	SPACE RUN
0111 1	1
0112 Σ	SUM
0113 3	3
0114 RTN	$ RTN

Details

IV

This program is run as shown below.

1. Press QUE to initialize.

2. Enter each word by pressing ALPHA, "WORD", ALPHA, e_1. Up to 5 characters can be submitted per word. Each word must be left justified.

3. Press e_5 to run the program and print the results.

Each word is stored by exercising label e_1 that is in location 0102. Here, each word is taken and indirectly stored by register 3, beginning in register 51. These values are then analyzed and rearranged into alphabetical order by steps 0009 through 0078. This iterative process continues until all values are in order. Locations 0079 through 0101 sort out and print the words, now in alphabetical order.

Example: Alphabetize and list the following names: Fred, Bob, John, Bill and Syd.

Press	Display	Printer
QUE	51.	
ALPHA FRED ALPHA e_1	÷	
ALPHA BOB ALPHA e_1	0	
ALPHA JOHN ALPHA e_1	0	
ALPHA BILL ALPHA e_1	0	
ALPHA SYD ALPHA e_1	0	
e_5	1.	BILL BOB FRED JOHN SYD

V

GEN

Both p
The ca
gram c
both s
for ea
Refer

The c
480 p
ters o

Progr
instru

A ma
keybo
print
slot u
throu
mach

To r
cards
to th
there

V READING AND RECORDING MAGNETIC CARDS

GENERAL INSTRUCTIONS

Both program instructions and data register contents can be read from and recorded onto magnetic cards. The cards used with the SR-60A have two sides that function independently. This means that a program can be stored on one side of a card and data on the other, or programs on both sides, or data on both sides. Cards are marked with "Side A Program" along one edge and "Side B Program" on the other for easy identification. Space is provided on each side for writing the title of the stored program or data. Refer to Appendix B, Maintenance and Service for instructions in caring for magnetic cards.

The contents of 60 data registers or 480 program steps can be stored on each side of a card. Therefore, 480 program instructions could be recorded on one side of a card and the contents of the 60 data registers on the other.

Program memory is initially filled with null instructions that do nothing – they're just space holders. Null instructions remain in all locations that are not deliberately replaced by program instructions.

A magnetic card can be recorded or read by inserting it into the slot in the upper right-hand part of the keyboard as shown in the figure below. After the appropriate key sequence, position the card with the printed side up and the arrow on the desired side pointing toward the slot. Gently feed the card into the slot until it is engaged by the drive motor. Do not hold or restrict the card while it is being pulled through the calculator. After the motor stops, remove the card from the slot on the top surface of the machine.

To record (write) on a card, you must first place a square black tab (a sheet of these comes with your cards) in the outlined box at the end of the arrow of the side you wish to record. This is an indication to the calculator that you, for sure, want to write on this card – cancelling anything previously stored there.

Reading and Recording Magnetic Cards

V

Inserting a Magnetic Card

V

Reading and Recording Magnetic Cards

CARD READING

READING PROGRAM INSTRUCTIONS

Card programs are loaded using one of the following procedures. If you want all registers cleared and all flags reset, follow procedure 1; otherwise, follow procedure 2.

Procedure 1

1. Press **CLEAR ALL**. The message "PROMPTING DESIRED?" will appear in the display.

2. Press **YES**. The message "PUSH YES, LOAD CARD" is displayed.

3. Press **YES** again and insert the first card.

4. After the drive motor stops, remove the card. If the message "PUSH YES, LOAD CARD" is displayed, proceed to step 5. If not, the program begins running.

5. Press **YES** and insert the next card.

"PUSH YES, LOAD CARD" will be displayed following reading of each card until a null instruction is encountered or until there are not 480 program steps left. The calculator interprets a null instruction on a card to mean that there are no more cards in the program. This means that *the last card of a program should have at least one null instruction to tell the calculator there are no more cards.* If the last instruction of a program is in the last location of a card, the calculator expects to read another card and displays "PUSH YES, LOAD CARD". In this case, simply press **NO** and the necessary label table will be built.

In the event of an error in reading a card, the display will flash with a question mark and the program pointer will return to its position at the start of that card. The card may be read again by pressing **CLEAR, READ.**

Procedure 2

1. Press **RESET, READ**

2. Insert first card. When card has been read, the message "PUSH YES, LOAD CARD" will appear in the display and steps 3–5 of procedure 1 can be followed from this point.

This procedure allows you to choose the location in program memory where you wish to start program storage. This is done by pressing **GO TO**, the location number (address), and **READ**. Entering one card of a multicard program can be done this way. However, storage may not begin at exactly an arbitrarily chosen memory location. The reason is that the calculator stores instructions internally by register with 8 instructions per register. Therefore, storage will begin at the nearest multiple-of-eight location that does not exceed the entered address. For example, **GO TO, 0395, READ** will cause the first instruction on the card to be stored in location 392. Attempting to read closer than 480 locations from the partition results in a flashing display and the drive motor will not start. Loading a program using procedure 2 does not affect pending operations, data registers, fix-decimal control or rounding select. However, a number entry or result will be cleared during the read sequence.

Again, the end of a program is signified by a null instruction. If a null instruction is not found on a card and there are less than 480 locations between the end of that card and the partition, the display flashes after reading that card. If more instructions are to be stored, you must either repartition to provide at least 480 available locations or reposition the program pointer to read over part of what's already been read. See *Recording a Card* for all information on how to create cards for this purpose. When a program completely fills the program memory (the last card of the program contains no null instruction), press **LEARN, LEARN, QUE** so that the calculator will build the necessary table of labels and start the program running.

A program that contains less than 480 steps must still be read into a memory that is partitioned at 480 or beyond because a card side (480 steps) is the least amount that can be read at a time. If a program is exactly 480 steps long and the memory area is partitioned at 480, press **NO** in answer to "PUSH YES, LOAD CARD" in the display.

READING DATA

The contents of 60 data registers can be contained on each side of a card. Data is read into the data registers using the following procedure.

1. Press **CLEAR, 2nd READ** and insert the first card. The 60 values that are read are placed in registers 00-59.

2. Press **60, 2nd READ** and insert the second card or side. The 60 values from that card are stored in data registers 60-119.

RECORDING ON CARDS

RECORDING PROGRAM INSTRUCTIONS

To record on a magnetic card, a black self-adhesive tab (furnished with blank cards) must be placed on the square near the tip of the arrow corresponding to the side to be recorded. After recording, remove the black tab to prevent accidental recording, and properly mark the cards to indicate the sequence in which they were recorded. Recording a program is performed using the following procedure.

1. Press **RESET, WRITE** (drive motor starts).

2. Insert the first card, remove after drive motor stops.

3. Press **WRITE.**

4. Insert the next card, remove after drive motor stops.

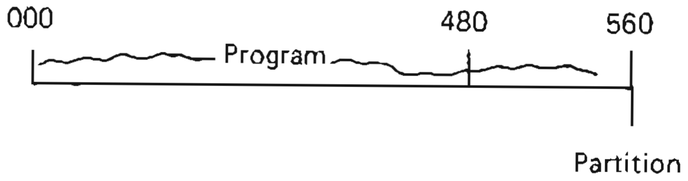
5. Repeat steps 3 and 4 for all cards.

If a flashing 0? appears in the display while attempting to record a card, press **CLEAR**, **WRITE** and reinsert the card. Each card begins writing from the current position (nearest register) of the program pointer. The program pointer advances 480 steps each time a card is written.

When attempting to write a card, there must be at least 480 locations between the position of the program pointer and the partition. If not, the display immediately flashes when **WRITE** is pressed.

You can begin writing almost anywhere in a program. Writing starts at the nearest multiple-of-eight location that does not exceed the location number where the program pointer is positioned. For instance, if the program pointer is on location 395, writing begins from location 392.

If a situation arises where the partition is not on a multiple of 480 (leaving the last segment of program with less than 480 steps), simply back up the program pointer to at least 480 steps ahead of the partition and write the last card. When this card is read, use the same technique to back up and read the card. For example, say the partition is at program location 560 and your program occupies most of this space.



Press **RESET**, **WRITE** to write the first 480 steps on one side of the card. Now back the program pointer up to 560 − 480 = 80 by pressing **GO TO 80** and press **WRITE** to write steps 80-559 onto the other side of the card. Attempting to enter the learn mode at this point causes the last location written (559) to flash in the display. This happens anytime you write the last location of program memory, because the next location to be written is normally displayed, but in this case location 560 is not in program memory.

RECORDING DATA

The contents of the data registers can be recorded on magnetic cards using the following procedure.

1. Press **CLEAR, 2nd, WRITE.**

2. Insert the first card, remove after drive motor stops.

3. Press **60, 2nd, WRITE.**

4. Insert the second card, remove after drive motor stops.

The number present in the display when **2nd**, **WRITE** is pressed determines the data register from which the calculator begins recording data. This permits data to be recorded starting with any data register.

A Appendix

INSTALLATION AND CHECKOUT

The procedures in this section should be followed for proper installation and checkout of the calculator. Any difficulties experienced after initial installation should be referenced to the Maintenance and Service section of this manual.

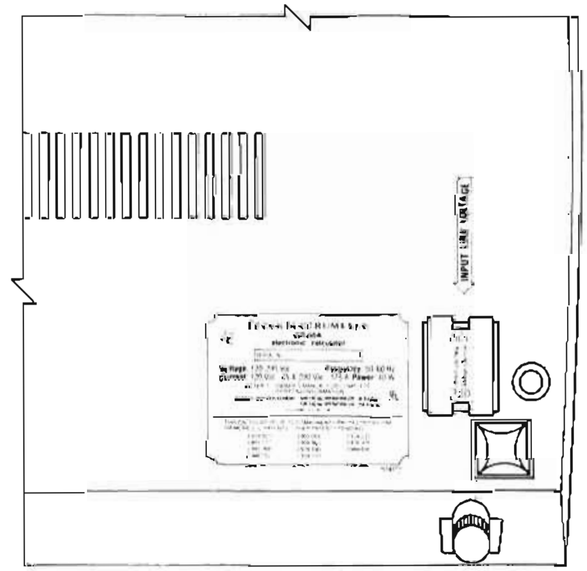
SETUP CONSIDERATIONS

CAUTION

Do not connect power cord until instructed to do so in these procedures.

Input Line Voltage Selection

The SR-60A will operate on either of the two standard ac voltages: 120 Vac or 240 Vac. The switch that selects the voltage to be used is located on the bottom of the calculator (see below) near the power switch. The calculator is supplied from the factory set up for 120 Vac operation.



Input Voltage Select Switch

For proper operation, when the input line voltage switch is set to 120 Vac, the measured line voltage should not be lower than 105 Vac or exceed 130 Vac. During 240 Vac operation, the measured line voltage should not be lower than 198 Vac or exceed 265 Vac. Power line frequency may be 50 or 60 hertz. Any questions or problems related to the input line voltage remaining in these limits should be referred to the building electrician or the local power company authorities.

Appendix A

INSTALLATION AND CHECKOUT

Fuse Selection

For proper overload protection, the calculator must be fused according to the input line voltage in use. The fuse holder is located on the back edge of the calculator below the recessed power connector. The fuse is removed from the holder by pressing in on the cap and rotating one-quarter turn counterclockwise. If the input line voltage is 120 Vac, the fuse should be a .5A (1/2A) Slo Blow. If the input line voltage is 240 Vac, the fuse should be a .25A (1/4A) Slo Blow. All machines are supplied from the factory with a .5A (1/2A) Slo Blow for 120 Vac.

Power Cord Installation

The power cord furnished with calculators is a six-foot detachable cord which will insert into standard 120 Vac/60Hz, three-conductor outlets (including safety ground). Optional power cords are available as accessories for calculators distributed outside the United States. Insert the power cord into the receptacle on the back edge of the calculator and connect the opposite end of the cord to the appropriate power source.

INITIAL CHECKOUT

Use the following initial checkout instructions only after reading and following the previous *Setup Considerations*.

Power On

Set the input power switch to the **ON** position. The display should show "PROMPTING DESIRED?".

The SR-60A will maintain memory contents during a power loss of up to two seconds when not printing. If power is not restored within two seconds, it should remain off for at least ten seconds to obtain a proper power-on indication. If a power loss duration of two to ten seconds occurs, set the calculator power switch off for ten seconds then set the power switch on again.

Functional Verification

1. Verify that printing paper is installed in the printing unit. If paper is not installed, refer to *Printing Paper Installation* in the Maintenance and Service section of this manual.

2. If "PROMPTING DESIRED?" is not displayed, press the **CLEAR ALL** key.

3. Locate the Basic Library magnetic card with the title "Diagnostic 1" on side A and "Diagnostic 2" on side B.

4. Press the **YES** key twice and insert the end of the magnetic card into the card-reader slot with the side A arrow pointing toward the slot as shown below. Gently feed the card into the slot until it is pulled through the calculator by the motor. Do not hold or restrict card travel while engaged by the motor.

INSTALLATION AND CHECKOUT

Loading A Magnetic Card

5. After the motor stops, remove the card. The printout should show "**DIAGNOSTIC 1" followed by "ENTER NO. REGISTERS" showing on the printout and the display. If after the motor stops, a flashing question mark appears on the display, repeat steps 2 through 5. If difficulty persists, refer to *In Case of Difficulty* in Appendix B.

6. Press **CLEAR**, **2nd X ≷ K**, **2nd Int x**, **X**, **100** (**1000** for expanded machines), **=** , **ENTER**. The printout will show the number of registers entered and the display flickers "TESTING" while the diagnostic program is running.

7. When display and printout show "END OF TEST", Diagnostic 1 has been successfully completed. If the diagnostic is unsuccessful, the printout will show the word "ERROR" one or more times with a number following the word. If ERROR 1 occurs, save the printout tape for the service representative.

8. Repeat steps 2 through 4 except insert the opposite end (side B) of the diagnostic card into the card-reader slot.

INSTALLATION AND CHECKOUT

9. After the motor stops, remove the card. The printout should show "**DIAGNOSTIC 2" and the display will show the word "TESTING" (flickering) while the diagnostic program is in progress. If after the motor stops, a flashing question mark appears on the display, repeat step 8. If difficulty persists, refer to *In Case of Difficulty* in Appendix B.

10. If the internal tests of Diagnostic 2 are not successful, the printout will show the word "ERROR" one or more times with a number following the word. If this occurs, save the printout tape and consult your local retailer.

11. When internal testing is successfully completed, printout should show "PRESS YES TO TEST DISPLAY, NO TO END" followed by II in all 20 character positions on the printout and display. The following four steps are a visual check of the display and printer. If the display or printout does not appear as described, consult your local retailer.

12. Observe that the displayed II appears in all 20 character positions.

13. Press the **YES** key one time. The character □ should appear in all 20 character positions of the display.

14. Press the **YES** key one time. The letter I should appear in all 20 character positions of the display.

15. Observe that the printout shows one line each (20 characters), II, □, and I. Due to variations in thermal printing paper, the dots on the printout will not all look alike with respect to darkness or contrast. If a specific area of dots appears lighter than others, press the **YES** key several times to see if the light dots remain in the same area. If the light area moves around, it is due to paper variations. If the light area is consistent, refer to *Using the Head-Cleaning Card* in the Maintenance and Service section. After cleaning the printheads, repeat the test by pressing the **YES** key several times. If printout is still too light in the same area, consult your local dealer.

16. Press the **NO** key. Printout and display should show "END OF TEST".

B Appendix

MAINTENANCE AND SERVICE

PRINTING PAPER INSTALLATION

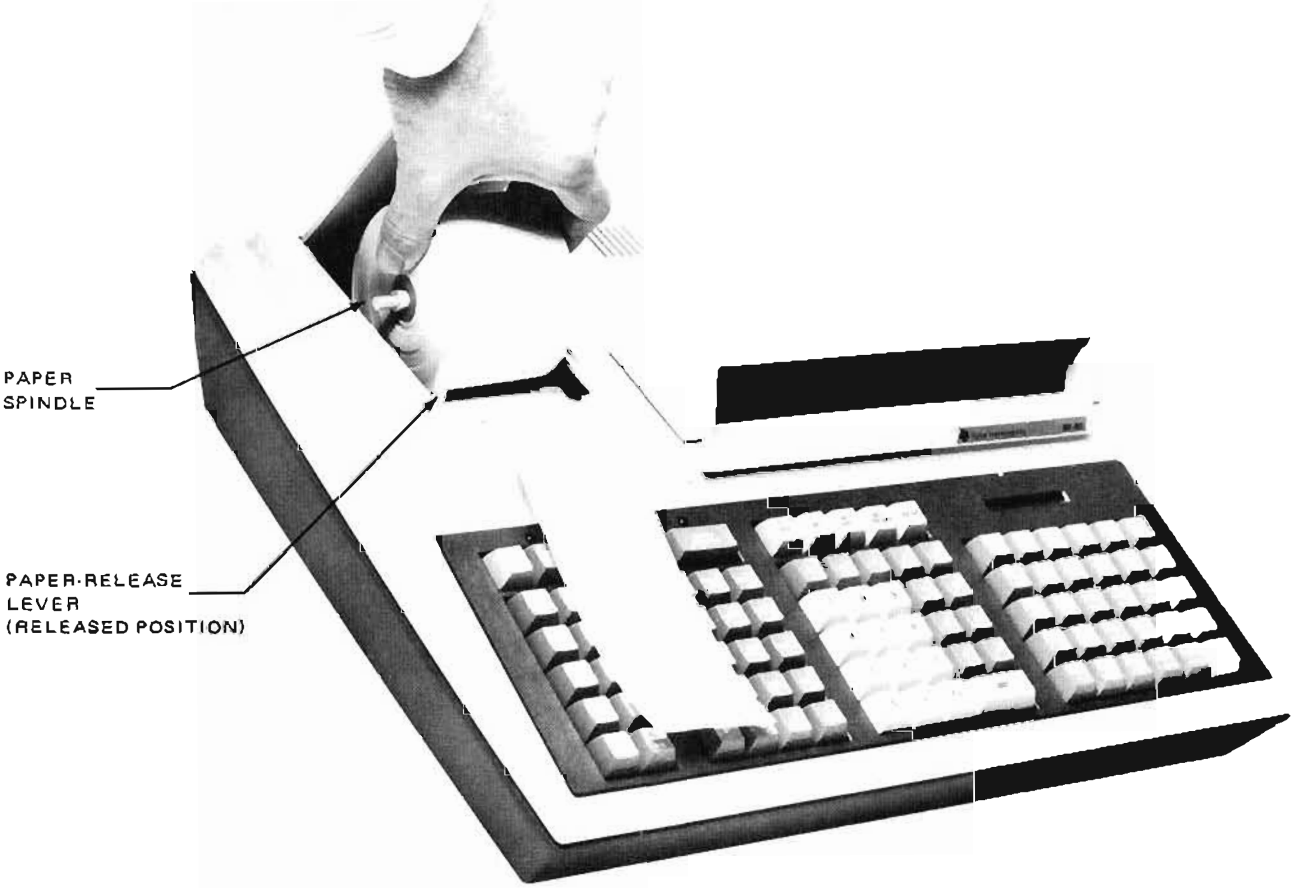
If you are performing calculations or running programs with lengthy printouts, check that sufficient paper remains to complete your calculations. A stripe will appear on the last few feet of the paper tape. When the printheads are resting directly against the rubber roller, the friction may be too great for the drive motor to overcome. When this occurs, a clicking noise will be heard when attempting to print or advance paper, but the rubber roller will not turn. This is a normal action and will not harm the printer mechanism unless continued for an extended period of time. A new roll of paper should be installed as soon as possible.

Follow these steps to install new paper:

1. Lift the cover over the printer to access the paper compartment.

2. The paper spindle is held in position by spring clips at the sides of the paper compartment. Grasp the paper spindle with the remaining paper roll and firmly pull upward to remove from paper compartment.

3. If paper is engaged by the rubber roller and printheads, pull the paper-release lever to the released position as shown below. The paper may now be gently pulled from the printer.



MAINTENANCE AND SERVICE

4. Insert the paper spindle in the new roll of thermal printing paper and position over the paper compartment as shown in the previous figure. Important: paper must unroll from bottom of the roll.

CAUTION

Use only TP-30250 thermal printing paper. Other papers may damage the calculator and void your warranty. Contact your retailer for paper.

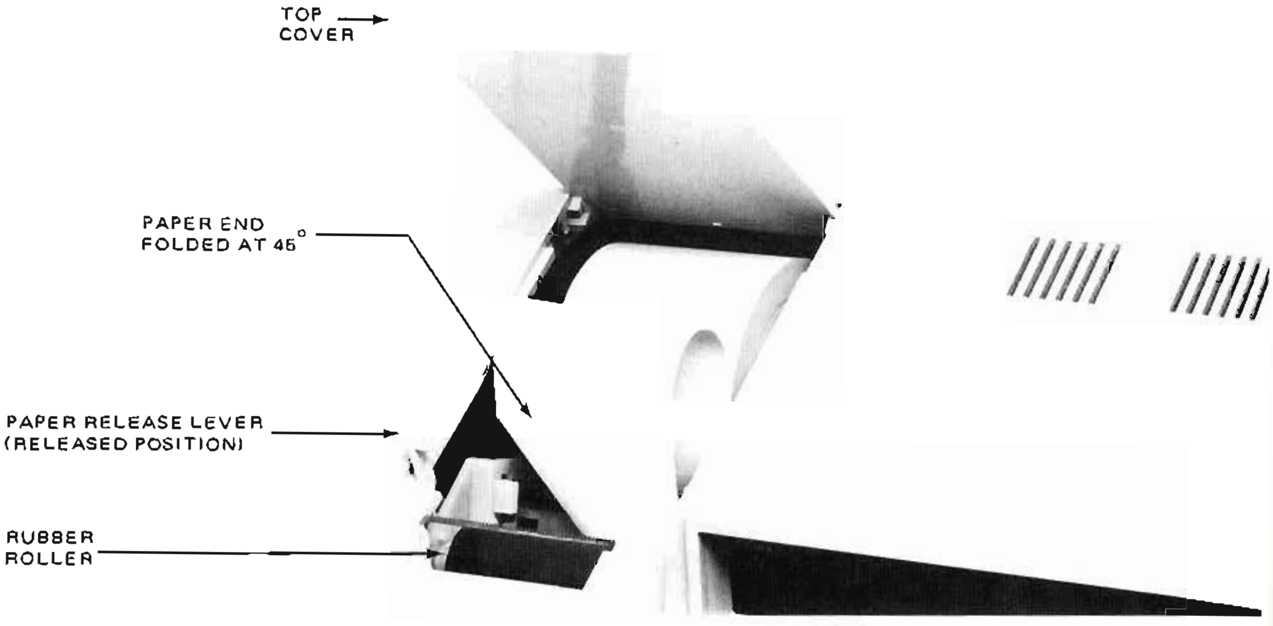
5. Firmly press the new roll of paper and paper spindle down into the paper compartment until the paper spindle snaps into place.

6. Fold the end of the paper tape to form a 45 degree angle as shown below.

7. Lift the paper-release lever to the released position and insert the point of the folded paper under the rubber roller.

8. Press the **PAPER ADV** key for the drive motor to pull the paper through the printer. When the point of the paper appears above the printheads, return the paper-release lever to the normal position. Hold the **PAPER ADV** key down until the folded portion of the paper is completely through the printer.

9. Hold the end of the paper slightly forward and close the top cover. Printer is now ready for normal operation.



Printing Paper Installation

MAINTENANCE AND SERVICE

CARING FOR THE PRINTER

Since the rubber roller is basically the only moving part, the printer requires a minimum amount of routine servicing, except for paper replacement. Occasionally, foreign particles may collect on the printheads causing digits or portions of digits to be faded on the printout. This type of problem is evident by a continuous faded streak which appears in the same physical position on each printed line. The following steps will normally correct this problem:

1. Cut an eight-inch length of standard bond paper with a width of two and one-half inches.

2. Move the paper-release lever to the released position as indicated in the previous diagram and gently pull the thermal printing paper out of the printer by manually turning the roll of paper.

3. Install the length of bond paper in place of the normal printing paper. Bond paper is normally rigid enough to permit pushing it through the printer without using the **PAPER ADV** key. Return the paper-release lever to the normal position.

4. Load Diagnostic 2 card as instructed in the Basic Library Manual. When the diagnostic program halts with the Roman numeral II in all 20 digits of the display, press the **YES** key 10 to 15 times, then press the **NO** key. The abrasive action of the bond paper cleans the printheads as should be evident by the faint print trailing to blank paper as the **YES** key is successively pressed.

5. Remove the bond paper from the printer and reinstall the thermal printing paper as prescribed at the beginning of this section.

If the above procedure does not remedy the faded printout problem, contact your service representative.

CARING FOR MAGNETIC CARDS

The 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 contaminant 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 will require repairs. The following simple instructions are important to assure maximum life of the magnetic cards.

1. Handle a card by its edges when possible.
2. Keep the cards away from magnetic fields and sharp objects that could scratch the oxide coating.
3. Keep the card in the furnished holder or other protective container while the card is not in use.
4. If a card is contaminated, clean it immediately.

Cleaning Cards

Contaminated cards may be cleaned easily without using special cleaners or solvents. **Petroleum based fluids or alcohol should not be used under any circumstances to clean cards.** Dust and foreign particles should be removed from a card with a soft brush or a dry soft cloth. Other forms of contamination may be washed from the card with warm water and a small amount of **mild** liquid detergent (not soap). Immediately rinse the card and dry with a soft cloth.

Writing 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 office 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.

CAUTION

> Heavy smudges or writing within 1/2 inch of the edge of the card can cause improper read or record operation.

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 buildup 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* instructions should normally be used as the guide for when the head-cleaning card may be used to remedy a difficulty. To use the card, press READ, then insert the card into the card-reader slot of the calculator as you would a regular card, and let the drive motor pull the card through the calculator. Press

CLEAR if the display flashes after using the card. The head-cleaning card should be used sparingly and no more than one time per difficulty.

IN CASE OF DIFFICULTY

In the event that you have difficulty with your calculator, the following instructions will help you analyze the difficulty. You may be able to remedy the difficulty without contacting a service representative. If the suggested remedies are not successful, see *If You Need Service Information* at the end of this section.

If one of the following symptoms appears while operating the calculator with auxiliary equipment connected, disconnect the cable to the equipment from the back of the calculator. If the symptoms disappear, refer to the manual for the auxiliary equipment.

CAUTION

> None of the suggested remedies require the calculator to be disassembled. The modules in the calculator must only be serviced by a qualified service representative.

1. **Display is blank for no obvious reason and entries are not accepted.**

   a. Press the CLEAR ALL key.

   b. If display turns on, calculator may have been in an endless program loop or the QUE instruction was executed without a prompting message in the alpha display register.

   c. A power interruption of 2 to 10 seconds may result in a blank display. Turn the power off for at least ten seconds and turn back on.

   d. If the display does not turn on, check the ac power input, check the fuse, check the position of the input voltage switch on the bottom of the calculator.

2. **Display flashes after loading a magnetic card.**

   a. The calculator has detected a reading error. Repeat the card loading procedure.

   b. If difficulty continues, try loading other cards. If other cards read properly, check the first card for physical defects, contamination or an altered program. Clean, replace or rerecord card as necessary.

   c. If other cards to not read properly, use the head-cleaning card one time – refer to *Using the Head-Cleaning Card*. Try loading the program card again.

   d. You may be trying to read too close to the partition.

3. **Display flashes after recording a magnetic card.**

   a. Black tabs are missing or incorrectly positioned over the write-protect windows on the magnetic card. Attempt to rewrite program.

4. **A program which has been loaded from a prerecorded card does not run properly.**

   a. Check contents of program memory against program listing for that program.

   b. If an incorrect instruction is found in program memory, reload the card. If incorrect instruction is still present, check card for physical defects. Rerecord or replace card as necessary.

   c. If an incorrect instruction is not found in program memory, perform Diagnostic 1 and Diagnostic 2 from the Basic Library. If an error code is printed when running either diagnostic, reload the card and check for same error code.

5. **Display flashes a message or result with a question mark when running program.**

   a. Entries were made which were outside the functional limits of the program, causing an overflow, underflow or error condition while the program was running.

   b. Incorrect key or key sequences were used to respond to a prompting message. Refer to the user instructions for the program.

6. **Calculator displays incorrect results.**

   a. Perform Diagnostic 1 and Diagnostic 2 from the Basic Library.

   b. If an error code is not displayed when running either diagnostic, check for invalid key sequence.

   c. If an error code appears, reload diagnostic and check for same error code.

7. **Printout has missing digits or parts of digits.**

   a. Perform Diagnostic 2 from Basic Library.

   b. If missing digits are confirmed, refer to *Caring for the Printer* in this section.

8. **Printer chatters when the PAPER ADV key is pressed.**

   a. Out of paper. Refer to Printing Paper Installation in this section.

9. **Printer works mechanically, but no digits are printing.**

   a. Thermal printing paper is installed with wrong side of paper against printheads.

MAINTENANCE AND SERVICE

IF YOU NEED SERVICE INFORMATION

If you have questions or need assistance with your calculator, contact the retailer from which the unit was purchased. He will either provide the necessary service or provide you with the name and address of the nearest authorized servicing retailer.

C

Appendix

OVERFLOW, UNDERFLOW AND ERROR CONDITIONS

A number of different situations result in a flashing display with a question mark, signaling an overflow, underflow or error condition. These conditions and the quantity flashed relative to keyboard calculations are summarized here.

Underflow and Overflow — When a number entry or calculation results in a non-zero quantity whose magnitude is less than 1. X 10^{-99} an underflow condition exists and the display flashes 1. −99?. Similarly, if a magnitude greater than 9.999999999 X 10^{99} should occur, the display will flash 9.999999999 99? to indicate an overflow condition. If a data register is forced into underflow or overflow condition, the display will flash the current quantity with a question mark. When recalled, an overflow in a data register will cause a flashing display. An overflow condition will also appear if a number is divided by zero, if the reciprocal of zero is taken, or the natural or common logarithm of zero is taken. Zero divided by zero will show a flashing 1 with a question mark.

Function Argument Outside of Range — Several functions have certain restrictions placed on their arguments in addition to those imposed by the underflow conditions. The functions, invalid arguments, and the error indications are indicated below. Note that a flashing display is accompanied by a question mark.

Function	Invalid Argument	Quantity Flashed
$x!$	$x < 0$ or non-integer	$(\text{Int } \lvert x \rvert)!$
$\sqrt{x}$	$x < 0$	$\sqrt{\lvert x \rvert}$
$\ln x$	$x < 0$	$\ln \lvert x \rvert$
$\log x$	$x < 0$	$\log \lvert x \rvert$
$\sin^{-1} x$	$\lvert x \rvert > 1$	x
$\cos^{-1} x$	$\lvert x \rvert > 1$	x
y^x	$y < 0$	$\lvert y \rvert^x$
$\sqrt[x]{y}$	$y < 0$	$\sqrt[x]{\lvert y \rvert}$
$\sqrt[x]{y}$	$x = y = 0$	1.

Exceeding Capacity of Processing Registers — The internal processing registers that hold pending operations and numbers in mathematical operations can accommodate up to nine pending open parentheses or up to ten pending operations with eleven pending operands. An attempt to enter more than nine pending open parentheses or ten pending operations will cause the display to flash the current quantity with a question mark.

Appendix C

OVERFLOW, UNDERFLOW AND ERROR CONDITIONS

Illegal Operation Sequences – Various keystroke sequences are considered by the calculator to be illegal. Pressing any two of these operations, +, –, X, ÷, y^x, $\sqrt[x]{y}$, and Δ%, sequentially will cause the display to flash the current quantity with a question mark. Also, following any of these operations with = or), or leading with (, will produce the same results. If both numbers connected by one of the above operations are the same number, the CE key may be used to reestablish the first number as the second number.

Example: $5 \times (5 + 5^5) = 15650$

Press	Display
5 × (	5.
CE +	5.
CE y^x	5.
CE =	15650.

Clearing and Removing Error Conditions – The most practical method of removing an error condition (flashing display and question mark), is to use the CE key. Pending operations and the displayed number are not affected. Note that a flashing display does not restrict any of the calculator functions as it is strictly an indication.

Pressing the **CLEAR** key removes an underflow, overflow, or error condition from the display. An overflow condition in a data register can be cleared with the **CLEAR MEM** key (that clears all data registers) or by storing another number in that register. The **CLEAR ALL** will, of course, clear everything, but it is not suggested as the normal method of clearing an underflow, overflow or error condition.

When an error indication is displayed resulting from too many pending parenthesis or operations, pressing CE, = will evaluate the total entry up to the key entry which caused the error conditions. When an error condition is displayed resulting from an illegal operation sequence, press CE and reenter the operation that caused the error condition to use the displayed number as the first and second operands. An illegal operation sequence involving the equals key cannot be saved with the CE key, reenter the problem.

D Appendix

CALCULATOR SPECIFICATIONS

GENERAL

Dimensions

Length:	17 inches
Width:	14.7 inches
Height:	5.5 inches

Weight

16 lbs

Power Requirements

Voltage:	105–130 Vac/198–265 Vac
Current:	.35A/.175A
Frequency:	50–60 Hz
Power:	40 W

Environmental Parameters

Temperature:	Operating 5°C to 40°C (41°F to 104°F)
	Storage –40°C to 70°C (–40°F to 158°F)
Relative Humidity: (maximum)	Operating 85%
	Storage 95%

Display Characteristics

Maximum Characters:	20
Character Format:	5 X 7 dot matrix
Type of Display:	Light-emitting diode (LED)

Printer Characteristics

Maximum Characters per line:	20
Character Format:	5 X 7 dot matrix
Type of Printer:	Thermal electronic
Type of Paper:	2.5 inch thermal

Printout Functions

Print, trace, paper advance, list (program), list (data registers). Lists program alphanumerics as characters to be printed, not the associated key.

CALCULATING FEATURES

Calculating Digits

Internal *AOS:*	12 digits plus sign and 2 exponent digits plus sign
Display or print:	10 digits plus sign and 2 exponent digits plus sign
Numerical range:	±1 X 10^{-99} to ±9.999999999 X 10^{99}
Data memory and algebraic functions:	13 digits plus sign and 2 exponent digits plus sign

Appendix D

CALCULATOR SPECIFICATIONS

Format:	Standard display or scientific notation
Format Options:	Fixed-length decimal fractions (Fix-decimal) – 0 to 8 digits
	Limited Precision
	Rounding – Round up, round down or round off with any integer weighting ratio.
Overflow, underflow or error indication:	
	Display – Flashing with question mark
	Printer – Prints question mark

Basic Functions

Entry format:	Algebraic
Completion order:	Δ%, y^x and $\sqrt[x]{y}$, X and ÷, + and –, = (special functions and conversions do not affect completion order).
Processing levels:	Up to 10 pending operations with 11 pending operands.
Parenthesis levels:	Up to 9 pending parentheses

Special Functions

Trigonometric:	Sine, cosine, tangent and inverses (degrees or radians)
Hyperbolic:	Sine, cosine, tangent and inverses
Logarithms:	Common, natural, 10^x, e^x
Misc:	Reciprocal, factorial, square, square root, integer, percent, percent difference, constant, pi
Conversions:	Degrees to radians, degrees-minutes-seconds to decimal degrees, polar to rectangular and their inverses

Data Registers

Number of registers:	Variable – up to 330 on basic unit (780 and 990 for memory options 2 and 3)
Independent functions:	Store, recall, sum, subtract, product, divide, exchange for any register DSZ for register 0
Other features:	Indirect addressing and clear memories. Alphanumeric information can be stored in data registers

D Appendix

CALCULATOR SPECIFICATIONS

PROGRAMMING FEATURES

Program Memory

Size:	Variable – up to 2640 for basic unit (6240 and 7920 for memory options 2 and 3)
Possible labels:	152
Subroutine levels:	12
Number of flags:	10

Program Functions

Learn, run, go to, halt, step, back step, insert, delete, reset, label, pause, subroutine, return, que, indirect, if error*, if positive*, if zero*, set flag*, test flag*, read, write, auxiliary and alpha. Five direct user-defined keys e_1 through e_5 and ten second-function user-defined keys e_6 through e_{15}. Alpha capabilities for labeling and prompting.

*Sense of operation reversed by **2nd** key prefix.

Alphanumeric characters

Alpha:	A through Z (Capitals only) and space
Punctuation:	Period, question mark, comma, apostrophe, parenthesis, dash
Symbols:	Dollar ($), degree (°), slash (/), asterisk (*)

Other symbols and characters are available by using the mathematical and function keys.

Magnetic Card

Card size:	10.5 × 2 inches
Card format:	Two sides
Capacity:	480 program steps per side or contents of 60 data registers per side.
Write protection method:	Optoelectronic. A black self-adhesive tab must be placed over write-protect window to record program or data register contents on a card.

E Appendix

INTERFACE AND EXPANSION CAPABILITIES

There are two memory options available for the SR-60A which may be added at any time. These options greatly expand the internal capacity of the calculator.

Option Name	Program Steps	Data Registers
MEMOPT2-60	Up to 6240	Up to 780
MEMOPT3-60	Up to 7920	Up to 990

Note that a MEMMODKIT2-60A kit may be required for the older SR-60 (not the SR-60A) calculators to permit installation of MEMOPT2-60 or MEMOPT3-60. If you do not know which memory option your 60A calculator contains, the following key sequences will isolate the size of the memory.

Turn on calculator, press NO, **2nd** x ≷ K

Display Contents	Memory Type
1919.99	Basic Unit
3839.309	MEMOPT2-60
5759.429	MEMOPT3-60

References to an "expanded machine" in the manuals apply to the memory options and also applies to program addressing (4-digit addresses) for MEMOPT2-60 and MEMOPT3-60 and to the basic unit with more than 999 program steps. All guidelines for "short-form" program addressing and "indirect" program addressing with the "expanded machine" apply to all memory options.

IMPORTANT: The Basic Library and other specialized libraries originated by Texas Instruments are compatible with all memory options of the SR-60A.

The basic procedure for reading and recording magnetic cards is the same as described for the "expanded machine" in the manuals except more cards are required.

Remember that the starting address of the data register must be entered in the display when reading or recording each side of a data card.

PROGRAMMING TIPS

Note that the time it takes to exit the learn mode increases with the size of program memory. This time is used to build the necessary label tables so that the program can run much faster. It may be faster in some cases to use step and backstep than to exit the learn mode and go to a new location. Also, the insert and delete instructions require completion times comparable to exiting learn mode when working at the beginning of program memory. However, when working near the end of program memory, insert and delete will be completed very quickly. When possible, programs should be entered and debugged near the end of program memory in 480-step increments. For example, start entering a program segment in the last 480 steps of MEMOPT2-60 by pressing GO TO, 3360, LEARN. After the program segment is entered and debugged, exit the learn mode, press GO TO, 3360, WRITE to record the program segment on one side of a magnetic card. When this is accomplished for all program segments, the card sides can be read in the order you wish the composite program to be stored. Remember, a null instruction at any location on a card side will cause the calculator to terminate the card-reading sequence after that card. Only the last card side to be read should contain a null instruction. The last 480 steps of MEMOPT3-60 is accessed by GO TO 5280.

INTERFACE CAPABILITIES

The SR-60A is designed to operate with auxiliary equipment such as the CPT *Selectric* Typewriter, a cassette storage system (single or dual) and the RS-232C interface. These connect through the peripheral interface that attaches to the connector on the back of the calculator. Through this interface unit, one or more of these peripheral units may be controlled by the **AUX** key on the SR-60A keyboard.

Index

Index

(continued)

Index

(continued)

Index

(continued)

(continued)

TEXAS INSTRUMENTS
INCORPORATED
DALLAS TEXAS