

# TEXAS INSTRUMENTS

## PROGRAMMABLE

# TI-95

## PROGRAMMING GUIDE





# TEXAS INSTRUMENTS

# **TI-95**

## **PROGRAMMING GUIDE**

**Manual  
developed by:**

The staff of Texas Instruments  
Instructional Communications and  
Design Communications  
TI Corporate Design Center

Michael T. Keller  
Chris M. Alley  
David Thomas  
De Warden  
Joseph L. Willard

**With  
contributions  
by:**

Linda Ferrio  
Art Hunter  
Robert A. Pollan  
Jacquelyn F. Quiram  
Robert E. Whitsitt, II

---

# DATAMATH CALCULATOR MUSEUM

---

## Important

Texas Instruments makes no warranty, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding these programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability to Texas Instruments, regardless of the form of action, shall not exceed the purchase price of this calculator. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the user of these programs or book materials by any other party.

Use this list if you know the function you want to perform but you need a reminder of the key sequence. The page reference tells where to find additional information about the function. This list includes programming-related functions only; scientific-calculator functions are discussed in the *TI-95 Users Guide*.

General Functions	Key Sequence	Page
Learn Mode	<b>LEARN</b>	1-6, 1-18
Toggle Program Counter	<b>2nd</b> [PC]	2-6, 2-15
Clear Program	<b>2nd</b> [CP]	1-6, 1-18
Run Program	<b>RUN</b>	1-10, 1-20, 8-8, 8-33
Check Partitioning	<b>INV</b> <b>2nd</b> [PART]	7-8, 7-14
Partition User Memory	<b>2nd</b> [PART]	7-9, 7-14, 7-15, 7-16
Halt	<b>HALT</b>	1-5, 1-20, 2-15
Break	<b>BREAK</b>	2-10, 2-15
Continue Execution	<GO>	2-10, 2-15
No Operation	<b>2nd</b> [NOP]	1-19
Pause	<b>2nd</b> [PAUSE]	2-13, 2-15
Restore Menu/Message	<b>OLD</b>	3-17, 4-21, 4-29
Assemble Program	<b>2nd</b> [ASM]	4-25, 4-29
Disassemble Program	<b>INV</b> <b>2nd</b> [ASM]	4-25, 4-29
Indirect Addressing	<b>2nd</b> [IND]	6-3, 6-14
Program Editing Functions	Key Sequence	Page
Right	<b>→</b>	1-18
Left	<b>←</b>	1-19
Insert Instructions	<b>2nd</b> [INS]	1-14, 1-19
Delete Instruction	<b>2nd</b> [DEL]	1-15, 1-19
Alpha Functions	Key Sequence	Page
Delete Character	<b>ALPHA</b> <DEL>	3-11, 3-18
Insert Characters	<b>ALPHA</b> <INS>	3-11, 3-18
Right	<b>→</b>	3-10, 3-18
Left	<b>←</b>	3-10, 3-18
Set Cursor to Column	<b>ALPHA</b> <COL>	3-10, 3-19
Merge Number	<b>ALPHA</b> <MRG>	3-12, 3-19
Recall Alpha	<b>ALPHA</b> <--> <RCA>	3-9, 3-19, 3-20
Store Alpha	<b>ALPHA</b> <--> <STA>	3-9, 3-20
Character Code	<b>ALPHA</b> <--> <CHR>	3-7, 3-20
Toggle Lowercase Lock	<b>ALPHA</b> <--> <LC>	3-6, 3-20

(continued)

Listing Functions	Key Sequence	Page
List Program	<b>LIST</b> <PGM>	1-12, 1-21
List Labels	<b>LIST</b> <LBL>	4-24, 4-26
List Status	<b>LIST</b> <ST>	2-9

Transfer Functions	Key Sequence	Page
Label Segment	<b>2nd</b> <b>[LBL]</b>	4-5, 4-26
Go To Label	<b>2nd</b> <b>[GTL]</b>	4-8, 4-26
Go To	<b>INV</b> <b>2nd</b> <b>[GTL]</b>	4-10, 4-26
Subroutine Label	<b>2nd</b> <b>[SBL]</b>	4-13, 4-27
Subroutine	<b>INV</b> <b>2nd</b> <b>[SBL]</b>	4-13, 4-27
Return	<b>2nd</b> <b>[RTN]</b>	4-11, 4-27
Define Function Key	<b>2nd</b> <b>[DFN]</b>	4-16, 4-28
Clear All Function Keys	<b>2nd</b> <b>[DFN]</b> <b>[CLEAR]</b>	4-23, 4-28
Clear One Function Key	<b>2nd</b> <b>[DFN]</b> <b>Fr</b> <b>[CLEAR]</b>	4-23, 4-29

Test Functions	Key Sequence	Page
If Greater Than	<b>TESTS</b> <IF>>	5-4, 5-25
If Less Than or Equal to	<b>TESTS</b> <b>INV</b> <IF>>	5-4, 5-25
If Less Than	<b>TESTS</b> <IF<>	5-4, 5-25
If Greater Than or Equal to	<b>TESTS</b> <b>INV</b> <IF<>	5-4, 5-25
If Equal to	<b>TESTS</b> <IF = >	5-4, 5-25
If Not Equal to	<b>TESTS</b> <b>INV</b> <IF = >	5-4, 5-25
Decrement and Skip on Zero	<b>TESTS</b> <DSZ>	5-7, 5-26
Decrement and Execute on Zero	<b>TESTS</b> <b>INV</b> <DSZ>	5-7, 5-26
Yes/No Response	<b>TESTS</b> <Y/N>	5-11, 5-26

Flag Functions	Key Sequence	Page
Clear Flags	<b>FLAGS</b> <CLR>	5-14, 5-27
Set Flag	<b>FLAGS</b> <SF>	5-14, 5-27
Reset Flag	<b>FLAGS</b> <RF>	5-14, 5-27
Test If Flag Set	<b>FLAGS</b> <TF>	5-14, 5-27
Test If Flag Reset	<b>FLAGS</b> <b>[INV]</b> <TF>	5-14, 5-27

File Functions	Key Sequence	Page
Load File	<b>FILES</b> <GET>	8-10, 8-16, 8-27, 8-30
Save File	<b>FILES</b> <PUT>	8-6, 8-12, 8-26, 8-29
Delete File	<b>FILES</b> <DF>	8-20, 8-28, 8-32
Display Catalog	<b>FILES</b> <CAT>	8-18, 8-28, 8-31
Clear Directory	<b>FILES</b> <--> <CD>	8-21, 8-28, 8-32
Name Cartridge	<b>FILES</b> <--> <NAM>	8-5, 8-27, 8-31
Input/Output Functions	Key Sequence	Page
Write Tape File	<b>I/O</b> <TAP> <WRT>	9-9, 9-22, 9-24
Read Tape File	<b>I/O</b> <TAP> <RD>	9-12, 9-22, 9-24
Verify Tape File	<b>I/O</b> <TAP> <VFY>	9-12, 9-23, 9-25
Call I/O	<b>I/O</b> <CIO>	B-2, B-15
Key Wait	<b>I/O</b> <KW>	B-22
System Functions	Key Sequence	Page
Store Byte	<b>FUNC</b> <SYS> <STB>	A-7
Recall Byte	<b>FUNC</b> <SYS> <RCB>	A-7
Assembly Language Subroutine	<b>FUNC</b> <SYS> <SBA>	A-15
Unformatted Mode	<b>CONV</b> <BAS> <UNF>	A-8

# Table of Contents

---

This book describes the programming functions of the TI-95 calculator. Before using this book, you should already be familiar with the scientific-calculator functions, described in the *TI-95 User's Guide*.

---

<b>Chapter 1:</b>	Programming Keys Used in This Chapter . . . . .	1-2
<b>Working with</b>	Introduction . . . . .	1-3
<b>Programs on</b>	Entering a Program . . . . .	1-6
<b>the TI-95</b>	Running a Program . . . . .	1-10
	Listing a Program . . . . .	1-12
	Editing a Program . . . . .	1-14
	Reference Section . . . . .	1-18
<b>Chapter 2:</b>	Introduction . . . . .	2-2
<b>Using Calculator</b>	Using System-Menu Functions . . . . .	2-6
<b>Functions in</b>	Keyboard Commands versus Program Instructions . . . . .	2-9
<b>a Program</b>	Making a Program Wait for Numeric Input . . . . .	2-10
	Pausing a Program to Display Results . . . . .	2-13
	Reference Section . . . . .	2-15
<b>Chapter 3:</b>	Introduction . . . . .	3-2
<b>Displaying</b>	The Alpha Menu . . . . .	3-4
<b>Messages</b>	Creating an Alpha Message . . . . .	3-5
	Storing and Recalling Alpha Messages . . . . .	3-8
	Editing an Alpha Message . . . . .	3-10
	Merging Numbers with a Displayed Message . . . . .	3-12
	Using Alpha Messages in a Program . . . . .	3-14
	Reference Section . . . . .	3-17

<b>Chapter 4: Controlling the Sequence of Operations</b>	Introduction . . . . .	4-2
	Before Proceeding with this Chapter . . . . .	4-4
	Using Program Labels . . . . .	4-5
	Using Go To Label . . . . .	4-8
	Using Go To . . . . .	4-10
	Using Subroutines in a Program . . . . .	4-11
	Programming the Function Keys . . . . .	4-16
	Creating a Function-Key Menu . . . . .	4-18
	Restoring a User-Defined Menu . . . . .	4-21
	Clearing Function-Key Definitions . . . . .	4-22
	Listing Program Labels . . . . .	4-24
	Speeding Up Program Execution . . . . .	4-25
	Reference Section . . . . .	4-26
<b>Chapter 5: Using Tests in a Program</b>	Introduction . . . . .	5-2
	Using Comparison Tests . . . . .	5-4
	Using DSZ Tests . . . . .	5-7
	Using the YES/NO Test . . . . .	5-11
	Using Flags . . . . .	5-13
	Ways to Use Tests with Transfer Instructions . . . . .	5-16
	Using a Test to Exit a Loop . . . . .	5-22
	Reference Section . . . . .	5-25
<b>Chapter 6: Using Indirect Addressing</b>	Introduction . . . . .	6-2
	Indirect Data Register Operations . . . . .	6-4
	Indirect Transfer of Program Control . . . . .	6-8
	Other Indirect Operations . . . . .	6-13
	Reference Section . . . . .	6-14
<b>Chapter 7: Partitioning Memory</b>	Introduction . . . . .	7-2
	Why Change Memory Partitions? . . . . .	7-4
	Effects of Partitioning . . . . .	7-6
	Determining the Current Partitions . . . . .	7-8
	Partitioning from the Keyboard . . . . .	7-9
	Partitioning from Within a Program . . . . .	7-12
	Reference Section . . . . .	7-14

(continued)

---

## **Chapter 8: File Operations**

Introduction	8-2
Using the <b>FILE STORAGE</b> Menu	8-4
If Using a Constant Memory Cartridge	8-5
Saving a Program as a File	8-6
Running a Program File	8-8
Loading a Program File	8-10
Saving Data as a File	8-12
Loading a Data File	8-16
Listing the Catalog of a Directory	8-18
Deleting Files	8-20
Performing File Operations in a Program	8-22
Reference Section	8-24

## **Chapter 9: Cassette Tape Operations**

Introduction	9-2
Selecting a Cassette Recorder and Tapes	9-3
Guidelines for Good Recording	9-4
Connecting Your Recorder to the TI-95	9-6
Finding the Correct Volume Setting	9-7
Using the <b>TAPE STORAGE</b> Menu	9-8
Writing a File on Tape	9-9
Reading or Verifying a File on Tape	9-12
Examples of Tape Operations	9-16
Performing Tape Operations in a Program	9-18
Reference Section	9-20

## **Chapter 10: Developing Your Own Programs**

Programming Considerations	10-2
Planning Your Program	10-6

## **Appendix A: Advanced Memory Functions**

Introduction	A-2
Changing the System-protection Mode	A-4
Storing or Recalling a Single Byte	A-6
Using the Unformatted Mode	A-8
Internal Representation of Numeric Values	A-13
Accessing Assembly-language Subroutines	A-15

<b>Appendix B:</b>	Introduction .....	B-2
<b>Advanced</b>	The Parameters in a PAB .....	B-4
<b>Input/Output</b>	Command Codes .....	B-6
<b>Functions</b>	The Open Command .....	B-7
	The Read and Write Commands .....	B-10
	I/O Error Codes .....	B-11
	Performing an I/O Operation from a Program .....	B-12
	Example: Using CIO to Control a Peripheral .....	B-16
	Reading Keystrokes from a Program .....	B-22
<b>Appendix C:</b>	Character Codes .....	C-2
<b>Reference</b>	Program Codes .....	C-4
<b>Information</b>	Key Codes .....	C-6
	System Flags .....	C-7
	Instruction Mnemonics .....	C-10
	Useful Assembly-language Subroutines .....	C-14
	Summary of Field Instructions .....	C-16
	Memory Map .....	C-17
	System RAM Usage .....	C-18
	Index .....	C-21

A keystroke programming language is built into the TI-95. By using the features and capabilities of this language, you can reduce the amount of manual work required for repetitive problem-solving tasks.

---

## What Is Keystroke Programming?

Keystroke programming, in its simplest form, consists of storing a sequence of keystrokes in the calculator's memory. The key sequence that is stored is called a **program** and the process of storing the keystrokes is called **programming**. After storing a program in memory, you can perform the key sequence by **running** the program.

You can store virtually all the functions of the TI-95 in a program.

Besides the functions used for calculations, you can include special programming functions that enable your program to perform operations such as displaying messages, repeating key sequences, and making decisions.

## Why Write Your Own Programs?

The main advantages of writing a program are to save time, improve productivity, and avoid errors. Writing a program can be especially beneficial when:

- ▶ You need to solve a problem that requires an iterative process to arrive at a solution.
- ▶ You need to perform a lengthy calculation repetitively, using different data for each repetition.
- ▶ You need to change the sequence of steps in a calculation based on a condition or on the value of an intermediate result.
- ▶ You need to retain a keystroke solution for future use.

By writing a program for such tasks, you can reduce your workload to entering the numbers needed by the program. You do not need to reenter the keystrokes that are stored in the program. Your program can prompt you for the required information, control the sequence of steps, perform the calculations, and display and label the results.

---

## Programming Features

The TI-95 has features that make it easy to write programs. Some of these features include:

- ▶ 7200 bytes of user memory, partitioned as data registers, program steps, and file space. You can change the partitioning of this memory to suit your programming needs.
- ▶ An alphabetic display that shows program instructions in a readable, mnemonic form, rather than as a series of numeric codes that you must memorize or look up to interpret.
- ▶ An alpha mode that lets you display descriptive messages from within a program. You might use such messages to label the result of a calculation, request information, or indicate errors.
- ▶ Five user-definable function keys that can be labeled in the display. You can design your programs to define the function of each key.
- ▶ Storage and retrieval functions that let you save programs and data as files for later use. You can save files in the calculator's file space, in an optional Constant Memory™ cartridge, or on cassette tape using the optional CI-7 Cassette Interface Cable. Program files that are saved in the calculator's file space or in a Constant Memory cartridge can be executed directly from the file space or cartridge.

(continued)

**Using This Book** This book describes the functions used to program your calculator. The preliminary chapters cover functions basic to all programming tasks, such as entering and executing a program. Later chapters introduce more complex functions, with the final chapters discussing the most sophisticated functions of the calculator.

A reference section is included at the end of each chapter. This section can be beneficial in two ways. First, you can use it to review the information discussed in the chapter. Second, you can use it as a reference source when you want to look up details about the functions. Some of the detailed information provided in the reference sections is not repeated elsewhere. If you are already familiar with a keystroke programming language, you can use these sections to discover differences between the TI-95 and the calculator that you know.

Because each chapter builds upon the material discussed in earlier chapters, it is recommended that you read the chapters in the order presented.

The appendices provide information on manipulating the system memory of the calculator and using the input/output functions, as well as tables listing character codes, key codes, and instruction mnemonics.

---

## Notational Conventions

Many functions are incomplete without additional identifying data following the function. This data is referred to as a **field**. For example, to store a number in memory, you must follow the **[STO]** key with the address of the register in which you want to store the number. The register address is the field.

Notational conventions allow you to see at a glance whether a field is required by a function. The following notational conventions are used to represent fields in this book.

- ▶ An *n* character represents a digit (0–9) in a field. The number of *n* characters in the notation indicates the number of digits in the field. For example, **[INV]** **[2nd]** **[GTL]** *nnnn* indicates a four-digit field.
- ▶ An *X* character represents a letter in a field. Any letter from A through Z is valid for this field. For example, **[STO]** *nnn* or *X* indicates that **[STO]** can have either a three-digit numeric field or a letter field. In fields of this type, a lowercase letter is converted automatically to an uppercase letter.
- ▶ An *a* character represents an ASCII character (digit, letter, or punctuation sign) in a field. This type of field is referred to as an alphanumeric field. The number of *a* characters in the notation indicates the number of alphanumeric characters in the field. For example, **[2nd]** **[LBL]** *aa* indicates a two-character alphanumeric field. Alphanumeric fields can contain both uppercase and lowercase letters.
- ▶ An *h* character represents a hexadecimal digit in a field. The hexadecimal digits are 0 through 9 and A, B, C, D, E, and F (A through F are entered with **[2nd]** **[A<sub>H</sub>]** through **[2nd]** **[F<sub>H</sub>]**). The number of *h* characters in the notation indicates the number of hexadecimal digits in the field. For example, **<SBA>** *hhh* indicates a three-digit hexadecimal field.



# Chapter 1: Working with Programs on the TI-95

---

This chapter shows you how to enter, run, list, and edit a program.

---

<b>Table of Contents</b>	<b>Programming Keys Used in This Chapter</b> . . . . .	<b>1-2</b>
	Introduction . . . . .	1-3
	Entering a Program . . . . .	1-6
	Running a Program . . . . .	1-10
	Listing a Program . . . . .	1-12
	Editing a Program . . . . .	1-14
	Reference Section . . . . .	1-18

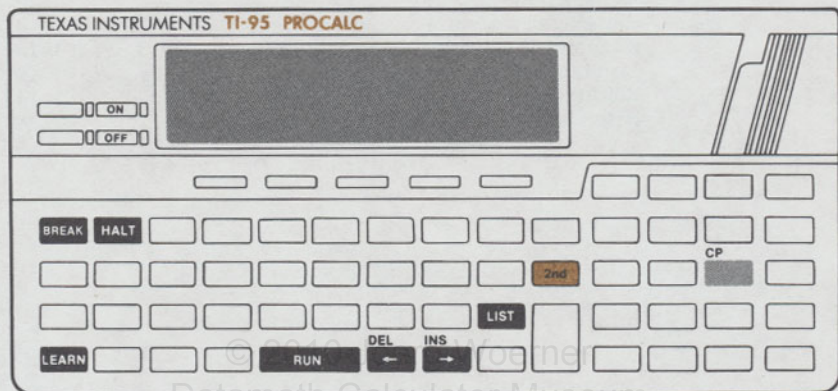
© 2010 Joerg Woerner  
Datamath Calculator Museum

## Programming Keys Used in This Chapter

---

The keys used to enter, run, list, and edit a program are shown in the figure below. Familiarize yourself with these keys and their locations on the keyboard.

---



As you enter a program, the calculator does not execute your keystrokes. Instead, it stores the keystrokes in a series of numbered memory locations called *program steps*. The number of program steps is controlled by the memory partitioning of the calculator.

## How Are Programs Stored?

Each program step is identified by a unique four-digit address. The first step is program address 0000. As a general rule, each keystroke is stored in a separate step. However, there are several exceptions.

- ▶ Second functions, such as **2nd** [x!], are combined into a single step. Only the actual operation, x! in this example, is stored.
- ▶ Menu functions, such as **CONV** <MET> <F-C>, are stored as a single step. Only the actual function, <F-C> in this example, is stored.
- ▶ Functions that require a field, such as **STO** 007, are stored in two or more steps—one for the instruction and one or more for the field.
- ▶ There are two inverse function key sequences in which **INV** combines with the function key sequence it precedes. These two functions, SBR and GTO, are discussed in Chapter 4.

## Arrangement of Program Steps

The following illustration can help you visualize the arrangement of program steps in memory. The “Address” column gives the number assigned to each program step.

Address	Program Step
0000	INV
0001	LOG
0002	+
0003	5
0004	x!
0005	=
0006	STO
0007	A
0008	HLT
0009	NOP

Any unused steps automatically contain a NOP (no operation) instruction.

### What Is an Instruction?

Once a function is stored in program memory, it is referred to as a program instruction. A program instruction is a key or key sequence that forms a complete function. For example:

- ▶ **y<sup>x</sup>** is an instruction.
- ▶ **2nd** **[x!]** is an instruction.
- ▶ **RCL** **A** is an instruction.
- ▶ **STO** **[+]** **007** is an instruction.
- ▶ **2nd** **[FIX]** **2** is an instruction.

### What Is a Mnemonic?

The symbol displayed by the calculator when you store a function in program memory is called a **mnemonic**. The mnemonic displayed for primary and second functions generally corresponds to the symbol shown on the keyboard. For some functions, the mnemonic is an abbreviated form of the key symbol.

Some examples of the mnemonics displayed for calculator functions are listed below.

Key Sequence	Mnemonic
<b>CLEAR</b>	CLR
<b>x</b>	*
<b>÷</b>	/
<b>+/-</b>	+/-
<b>x<sup>√t</sup></b>	x <sup>√t</sup>
<b>2nd</b> <b>[nPr]</b>	nPr
<b>INCR</b> <b>A</b>	INC A
<b>STO</b> <b>[+]</b> <b>007</b>	ST + 007

A complete list of the mnemonics of the calculator is given in Appendix C.

---

## How Are Programs Executed?

When you run a program, the calculator sets an internal pointer called a **program counter** to the first program step. It then executes the instruction stored at that location. After the proper action is accomplished, the program counter advances to the next instruction.

The calculator continues to advance the program counter and execute instructions until one of the following events occurs.

- ▶ You stop the program by manually pressing the **BREAK** or **HALT** key.
- ▶ The program is stopped by an instruction stored in the program, such as **BREAK** or **HALT**.
- ▶ The program counter reaches the end of program memory, which causes an error condition that stops execution. (Certain other errors also cause program execution to stop. See Chapter 5 for details.)

## Entering a Program

---

The learn mode of the calculator enables you to enter a sequence of keystrokes into program memory. Once entered, a program remains in memory until you clear it or replace it with another program.

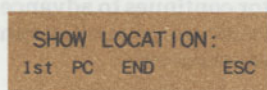
---

### Activating the Learn Mode

Before you can enter a program, the calculator must be in the learn mode. To activate the learn mode:

1. Press **[LEARN]**.

The following screen is displayed.



- |       |   |
|-------|---|
| 〈1st〉 | Positions the cursor to the first step in program memory.                                 |
| 〈PC〉  | Positions the cursor to the step specified by the current setting of the program counter. |
| 〈END〉 | Positions the cursor to the last instruction stored in program memory.                    |
| 〈ESC〉 | Clears the learn mode menu.   |

2. Select the option for the program location you want to display. The calculator displays the program counter (PC) on the second line of the display. The PC value corresponds to the address of the step marked by the cursor.

### Clearing the Program Memory

The key sequence **[2nd] [CP]** clears all keystrokes from program memory. When you clear program memory, the calculator fills all program steps with NOP (no operation) instructions. To prevent the accidental clearing of a program, the clear program function works only in the learn mode.

---

## Entering the Program

After selecting the learn mode, enter your program by pressing the desired sequence of keys. When a key (or key sequence) is pressed, the calculator enters that function into program memory and displays the mnemonic that represents the function.

As you enter your program, consider the following points.

- ▶ Enter the instructions in the order that you want them executed. (It is possible to change the order in which instructions execute by using transfer instructions. Transfer instructions are discussed in Chapter 4.)
- ▶ Include every instruction that you want executed. If you omit an instruction, your program will probably not run as intended.
- ▶ End your program with a **HALT** instruction. The **HALT** function instructs the calculator to stop program execution and return to keyboard operation.
- ▶ If you make a mistake in entering a program, you can correct it by using the editing keys discussed later in this chapter.

## Exiting the Learn Mode

After entering the keystrokes that make up your program, you must exit the learn mode before you can run the program.

To exit the learn mode, press **LEARN** again.

When the calculator exits the learn mode, it displays the contents of the numeric display register. Any alpha message that was in the display when the learn mode was entered initially is not redisplayed. (Alpha messages that you have created can be recalled, as explained in Chapter 3.)

(continued)

### Exiting the Learn Mode (Continued)

Exiting the learn mode does not affect the contents of program memory or change the location of the program counter. The program counter remains where it was last positioned, until you perform an operation that changes it.

### Example

To illustrate the process of programming the calculator, write a program to calculate the cube of a number. To make this calculation manually, you would:

- ▶ Enter the number to be cubed.
- ▶ Press  $\boxed{y^x}$  to specify the power function.
- ▶ Press 3 to specify the power.
- ▶ Press  $\boxed{=}$  to complete the calculation and display the result.

The last three actions in the list are the actions that you want to put into the program. Do not put the number to be cubed into the program, or you will have to change the program every time you want to cube a different number.

You write this program by entering the learn mode and duplicating the keystrokes in the list. The program design assumes that the number to be cubed is already in the display when the program is started.

**Example  
(Continued)**

To write the program, duplicate the keystrokes shown below.

Procedure	Press	Display
Activate learn mode	<b>LEARN</b>	SHOW LOCATION:
Display first step	<1st>	
Clear program memory	<b>2nd</b> <b>[CP]</b>	
Enter function	<b>y<sup>x</sup></b>	y <sup>x</sup>
Specify power	3	y <sup>x</sup> 3
Calculate result	<b>=</b>	y <sup>x</sup> 3 =
Stop execution	<b>HALT</b>	y <sup>x</sup> 3 = HLT
Exit learn mode *	<b>LEARN</b>	0.

\*The value displayed is the value that was in the numeric display register before you activated the learn mode.

The program is stored in the calculator. The procedure for running the program is described on the following pages.

## Running a Program

---

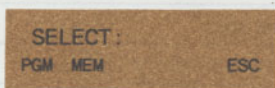
The advantage of entering a program into memory is that you can perform the same key sequence as many times as needed, without having to re-enter the program keystrokes.

---

### Procedure

To run the program currently stored in program memory:

1. Be sure the calculator is not in the learn mode.
2. Press **[RUN]** to display the following menu.



- |       |   |
|-------|---|
| <PGM> | Runs the program in program memory, starting at program step 0000.  |
| <MEM> | Enables you to run a program you have saved in the file space. (File space is discussed in chapter 8, "File Operations.") |

**Note:** If an optional software cartridge or 8K Constant Memory™ cartridge is installed, the name of the cartridge is displayed in the menu. See the "File Operations" chapter for details on running a program in a Constant Memory cartridge.

3. If your program requires you to enter a number into the display before starting the program, enter the number.
4. Select the menu option for the program you want to run.

The calculator runs the program. The **RUN** indicator is displayed while the program is running.

**Example**

Using the program written on page 1–9, calculate the cubes of 5 and 3.

Procedure	Press	Display
Display the menu	<b>RUN</b> SELECT:	
Calculate $5^3$	5 <PGM>	125.
Calculate $3^3$	3 <PGM>	27.

You only need to press **RUN** when <PGM> is not showing above the **F1** key. As long as <PGM> is visible, the program can be executed by pressing **F1**.

© 2010 Joerg Woerner  
Datamath Calculator Museum

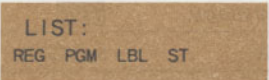
## Listing a Program

You can list the program currently in program memory by using the **LIST** <PGM> function. If you have a printer connected to the calculator, **LIST** <PGM> also prints the listing. The calculator displays the listing in groups. The address of the first instruction in the group is shown in the left side of the display.

### Listing the Program

To list the program currently in memory:

1. Be sure the calculator is not in the learn mode. Then press **LIST** to display the following menu.



LIST:  
REG PGM LBL ST

2. Press <PGM> to display the following menu.



START LISTING AT  
1st PC

- <1st> Begins at the first step in program memory.
- <PC> Begins at the address specified by the current setting of the program counter.

3. Select the point at which you want to begin the listing. Unless you pause or stop the listing as explained in the following sections, the calculator lists through the last step in the program.

### Controlling the Speed of the Listing

If you do not have a printer connected, the calculator pauses for one second before displaying the next group of program steps.

However, you can use the **→** key to control the speed of the listing.

- To pause the listing indefinitely, hold down the **→** key.
- To advance through the listing without the one-second pause, repeatedly press and release the **→** key.

## Stopping the Listing

To stop a program listing before it has finished, press and hold the **[BREAK]** or **[HALT]** key until the display returns to the list menu.

## Example

List the program that you entered on page 1–9. Before starting the listing, be sure the calculator is not in the learn mode.

Procedure	Press	Display
Select program listing	<b>[LIST]</b> <PGM>	START LISTING AT
Begin at first step	<1st>	0000 y <sup>x</sup> 3= HLT
Listing finishes		LIST:

Datamath Calculator Museum

## Editing a Program

You can edit a program by inserting, deleting, or replacing instructions. The calculator must be in the learn mode before you can edit the program. All changes occur at the location of the cursor. To move toward the beginning or end of a program, press  $\leftarrow$  or  $\rightarrow$ , respectively. Both keys repeat when held down for approximately one second.

### Inserting an Instruction

Pressing  $2^{\text{nd}}$  [INS] displays the INS indicator and places the calculator in the insert mode. In the insert mode, the instructions you enter are inserted into the program at the position of the cursor. The calculator remains in the insert condition until you cancel it by pressing  $\leftarrow$ ,  $\rightarrow$ ,  $2^{\text{nd}}$  [DEL], or [LEARN]. When you insert an instruction, the instruction at the current position of the cursor and all instructions that follow are shifted toward the end of program memory.

### Example

The following key sequence changes the program entered on page 1-9 so that it divides the cubed value by 2. The current program is  $y^x 3 = \text{HLT}$ . Insert “/2” before the “=” instruction.

Procedure	Press	Display
Activate learn mode	[LEARN] <1st>	$y^x$
Position cursor on =	$\rightarrow$ $\rightarrow$	$y^x 3 =$
Prepare for insert	$2^{\text{nd}}$ [INS]	$y^x 3 =$
Insert instructions	$\div$ 2	$y^x 3/2 =$
Exit learn mode	[LEARN]	0.

### Running the Example

Test your editing changes by running the new version of the program.

Procedure	Press	Display
Display RUN menu	[RUN]      SELECT:	
Enter the number	5	5
Display result	<PGM>	62.5

### Deleting an Instruction

To delete the instruction at the current position of the cursor, press **2nd** **[DEL]**. If the instruction has a field, the field is also deleted. All instructions following the deleted instruction are moved toward the beginning of program memory.

### Example

Delete the “/2” instructions from the sample program. This restores the program to its original form, which calculates the cube of a number.

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> <1st>	$y^x$
Move cursor to /	<b>→</b> <b>→</b>	$y^x$ 3/
Delete /	<b>2nd</b> <b>[DEL]</b>	$y^x$ 32
Delete 2	<b>2nd</b> <b>[DEL]</b>	$y^x$ 3=
Exit learn mode	<b>LEARN</b>	0.

### Running the Example

Test the program.

Procedure	Press	Display
Display <b>RUN</b> menu	<b>RUN</b>	SELECT:
Enter the number	5	5
Display result	<PGM>	125.

(continued)

### Replacing an Instruction

To replace an instruction, just “write over” the old instruction. For example, to replace a / with a \*, position the cursor on the / symbol and press **[X]**. Because / and \* each occupy one program step, no further change is necessary.

Watch that you do not leave an unwanted **[INV]** preceding the new instruction. Also consider whether the new instruction requires more steps than the instruction it replaces. If more steps are required, use the insert function to enter the extra keystrokes.

To replace the field of an instruction, you must reenter the entire instruction. For example, to replace **[STO] A** with **[STO] B**, place the cursor on **STO** and press **[STO] B**. The calculator does not permit you to position the cursor over a field, a feature that prevents accidental changes to the field.

If you replace an instruction that has a field with an instruction that occupies fewer program steps, the calculator stores NOP instructions in the extra steps. For example, if you replace **[STO] A** (which occupies two steps) with **[=]**, the calculator will place a **NOP** in the second step.

When you enter the first character or digit of a field, the calculator reserves enough program steps for the entire field. If you know that there is not enough space to enter the entire field without writing over existing instructions that you want to save, press **[2nd] [INS]** before you begin entering the field.

Use this section as a source of reference information on editing, running, listing, and saving a program.

**Example**

Change the sample program so that it raises the entered number to the fifth power instead of cubing it. The current program is  $y^x 3 = \text{HLT}$ . Replace the "3" with a "5."

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> <1st>	$y^x$
Position cursor on 3	<b>→</b>	$y^x 3$
Replace 3 with 5	5	$y^x 5 =$
Exit learn mode	<b>LEARN</b>	0.

**Running the Example**

Now test the program.

Procedure	Press	Display
Display <b>RUN</b> menu	<b>RUN</b>	SELECT:
Enter the number	5	5
Display result	<PGM>	3125.

Use this section as a source of reference information on entering, running, listing, and editing a program.

---

### Learn Mode

**[LEARN]**—Displays the learn mode menu or exits the learn mode, depending upon the calculator's state. When the learn mode is not active, pressing **[LEARN]** displays the learn mode menu. When the learn mode is active, pressing **[LEARN]** leaves the learn mode. Entering and exiting the learn mode clears the definitions (if any) of the function keys but does not clear the numeric display register.

**[LEARN] <1st>**—Sets the program counter to step 0000 and enters the learn mode at the beginning of program memory.

**[LEARN] <PC>**—Enters the learn mode at the current address of the program counter.

**[LEARN] <END>**—Sets the program counter to the last instruction in the program and enters the learn mode at that location.

### Clear Program

**[2nd] [CPI]**—Clears a program from memory. Clear program only operates when the calculator is in learn mode.



**[→]**—In the learn mode, the **[→]** key increments the program counter to the beginning of the next instruction, which is displayed in full, including any field. The key repeats if held down longer than one second.

Outside of the learn mode, the **[→]** key controls the speed of listing functions. Holding down the **[→]** key pauses the listing; repeatedly pressing it accelerates the listing. For details on the use of **[→]** in the alpha mode, refer to Chapter 3 of this guide.

**←**—In the learn mode, the **←** key decrements the program counter to the beginning of the previous instruction. The key repeats if held down longer than one second.

Outside of the learn mode, the **←** key removes the last digit of a numeric entry. For details on the use of **←** in the alpha mode, refer to Chapter 3 of this guide.

### Insert

**2nd** **[INS]**—Places the calculator in insert mode, permitting you to insert program instructions at the current location of the program counter (used only in the learn mode). Pressing **2nd** **[DEL]**, **←**, **→**, **[LEARN]**, or **[OFF]** cancels the insert mode.

The instruction (if any) in the highest-numbered program step in program memory is lost each time you insert an instruction.

### Delete

**2nd** **[DEL]**—Deletes the instruction, including any field, stored at the current location of the program counter (used only in the learn mode).

### No Operation

**2nd** **[NOP]**—Enters a no operation instruction into program memory. This function is valuable when you want to eliminate an instruction from a program without altering the addresses of any instructions in the program. No operation instructions are ignored during program execution.

A NOP instruction appears as a space character when the cursor is positioned over it in the learn mode.

(continued)

### Halt

**[HALT]**—Stops program execution without affecting the definitions (if any) of the function keys.

You can use **[HALT]** from the keyboard to stop a running program or a listing function. You can use a **[HALT]** instruction in a program to stop execution at the point in the program where the halt instruction is placed.

The instruction mnemonic for **[HALT]** is HLT.

### Run

**[RUN]**—As a keyboard command, **[RUN]** displays the program execution menu. As a program instruction, **[RUN]** enables an executing program to identify and execute another program. For details on the use of **[RUN]** as a program instruction, refer to page 8-34 of this guide.

**[RUN] <PGM>**—Runs the program currently stored in program memory, beginning with the first program instruction.

**[RUN] <MEM>**—Runs a program stored in the calculator's file space. For details on the use of this function, refer to page 8-8 of this guide.

**[RUN] <Directory>**—Runs a program stored in a software cartridge or Constant Memory cartridge. (A directory name appears in the menu only when a cartridge is installed in the calculator.) For details on the use of this function, refer to page 8-8 of this guide.

**Note:** Running a program does not clear calculations in progress. Therefore, if you decide to run a program before you complete a calculation, press **[CLEAR]** first to clear any pending numeric operations. You should also press **[CLEAR]** if you decide to rerun a program that was stopped in the middle of a calculation. To be certain that no pending numeric operations will affect your program's calculations, you can include a **[CLEAR]** instruction at the beginning of the program.

---

## List

**LIST**—Displays the list menu. Although all listing functions can be entered as program instructions, they are used primarily as keyboard commands. (Pressing **LIST** clears the numeric display register.)

Listings are normally displayed at a one-second rate, but they can be paused or accelerated using the **→** key. Listings stop automatically when the last item has been listed but can be cancelled manually by holding down the **BREAK** or **HALT** key.

If a printer is connected to the calculator, all listings are printed and displayed. The speed of the listing is determined by the printing rate of the printer.

**LIST** <REG>—Lists the contents of the data registers, starting with the register address in the numeric display register.

The list registers function is represented by the mnemonic LR in program memory.

**LIST** <PGM>—Lists the contents of program memory. When this key sequence is executed as a keyboard command, you are prompted to specify whether you want the listing to start at the beginning of program memory or at the current location of the program counter. When this function is executed in a program, the listing always starts at program address 0000.

The list program function is represented by the mnemonic LP in program memory.

**LIST** <LBL>—Lists labels used in program memory. For details on the operation of this function, refer to page 4-24 of this guide.

**LIST** <ST>—Lists status information about the calculator. For details on the operation of this function, refer to page 2-9 of this guide.



## Chapter 2: Using Calculator Functions in a Program

---

The *TI-95 User's Guide* describes the scientific-calculator functions and illustrates how to use them to perform manual calculations. This chapter discusses the use of those functions in a program. It also explains how you can stop a program to enter numbers or pause a program to display results.

---

Table of Contents	Introduction .....	2-2
	Using System-Menu Functions .....	2-6
	Keyboard Commands versus Program Instructions .....	2-9
	Making a Program Wait for Numeric Input .....	2-10
	Pausing a Program to Display Results .....	2-13
	Reference Section .....	2-15

© 2010 Joerg Wörmel  
Dataforth Calculator Museum

The scientific-calculator functions can be divided into two categories: functions accessed through conventional key sequences, such as  $\boxed{+}$  and  $\boxed{\text{INV}} \boxed{\text{SIN}}$ , and functions accessed through system-menu keys, such as  $\boxed{\text{CONV}} \langle \text{MET} \rangle \langle \text{G-L} \rangle$ . This section discusses the entry of conventional keys into a program.

### Storing Primary Functions

To store a key's primary function in a program, simply press the key for that function while the calculator is in the learn mode. For example, if you press  $\boxed{\text{SIN}}$  in the learn mode, the following mnemonic is displayed.



SIN

### Storing Second Functions

To store a second function in a program, press the  $\boxed{2\text{nd}}$  key followed by the key for that function. The mnemonic that is displayed represents the key's second function. For example, if you press  $\boxed{2\text{nd}} \boxed{\text{CMS}}$  in the learn mode, the following mnemonic is displayed.



CMS

### Storing Inverse Functions

To store an inverse function in a program, press the  $\boxed{\text{INV}}$  key followed by the key or key sequence for that function. For most inverse functions,  $\boxed{\text{INV}}$  is stored as a separate instruction. (Only two inverse functions do not store  $\boxed{\text{INV}}$  as a separate instruction. These are the transfer instructions  $\boxed{\text{INV}} \boxed{2\text{nd}} \boxed{\text{GTL}}$  and  $\boxed{\text{INV}} \boxed{2\text{nd}} \boxed{\text{SBL}}$ .)

For example, if you press  $\boxed{\text{INV}} \boxed{y^x}$  in the learn mode, the following mnemonics are displayed.



INV  $y^x$

---

### Storing Hyperbolic Functions

To store a hyperbolic function in a program, press the **[HYP]** key followed by the key or key sequence for that function. **[HYP]** is stored as a separate instruction.

For example, if you press **[HYP]** **[INV]** **[SIN]** in the learn mode, the following mnemonics are displayed.



HYP INV SIN

### Storing Functions with Fields


To store a function with a field in a program, press the key or key sequence for the function and then enter the field. The calculator groups the field characters together in the display.

For example, if you press **[STO]** 007 in the learn mode, the following mnemonic is displayed.



STO 007

You do not have to enter leading zeros when you store a numeric field if you use short-form addressing. For example, if you press **[RCL]** 7 **[x<sup>2</sup>]** in the learn mode, the following mnemonics are displayed.



RCL 007 x<sup>2</sup>

(continued)

**Example** Write a program to evaluate the following equation.

$$B = 3A^2 + 12A - 12$$

Assume that you enter the value of A into the display before you start the program. Because the equation uses A in two places, have the program store the value in a data register and recall it when it's needed again.

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> <1st>	
Clear program memory	<b>2nd</b> <b>[CP]</b>	
Store value of A	<b>[STO]</b> A	STO A
Square A	<b>[x<sup>2</sup>]</b>	STO A x <sup>2</sup>
Multiply by 3	<b>[x]</b> 3	STO A x <sup>2</sup> *3
Add	<b>[+]</b>	STO A x <sup>2</sup> *3 +
Enter second term	12 <b>[x]</b> <b>[RCL]</b> A	2 *3 + 12* RCL A
Enter third term	<b>[-]</b> 12	3 + 12* RCL A - 12
Obtain result	<b>[=]</b>	+ 12* RCL A - 12 =
Stop execution	<b>[HALT]</b>	RCL A - 12 = HLT
Exit learn mode	<b>LEARN</b>	

### Running the Example

To test the program, compute B for  $A = 5$ ,  $A = 11$ , and  $A = 1024$ .

Procedure	Press	Display
Display RUN menu	<b>RUN</b>	SELECT:
Calculate value	5 <PGM>	123.
Calculate value	11 <PGM>	483.
Calculate value	1024 <PGM>	3158004.

© 2010 Joerg Woerner  
Datamath Calculator Museum

## Using System-Menu Functions

---

When you include a function from a system menu in a program, the calculator stores only the function. The menu key is not stored in the program.

---

### Storing Menu Functions

To store a system-menu function in a program, press the menu key and make the selection you want. The calculator displays a mnemonic for the function you select, but the keystrokes that you use to access the menu function are not stored in the program.

For example, if you press **CONV** <MET> <G-L> in the learn mode, the following mnemonic is displayed.



G-L

The calculator stores the function to convert gallons to liters in the program, but it does not store **CONV** and <MET>.

### Restoring the Program Counter Display

Pressing a system-menu key displays the menu functions on the second line of the display. If the calculator is in the learn mode, the menu functions replace the program counter. To restore the program counter, press **2nd** [PC]. **2nd** [PC] alternately displays and erases the program counter.

- ▶ If the counter is in the display when you press **2nd** [PC], the calculator erases the counter.
- ▶ If the counter is not displayed when you press **2nd** [PC], the calculator displays the counter.

**Example**

The random-number function of the calculator generates numbers that are fractional values. By using the expression given below, you can generate random integers that lie between 1 and a given upper limit:

$$N = \text{INT}(L \times R\# + 1)$$

where N = the generated random integer, L = the selected upper limit, and R# = the random number produced by the calculator.

Using the formula given above, write a program that generates random integers between 1 and 6.

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> <1st>	
Clear program	<b>2nd</b> <b>[CP]</b>	
Enter upper limit	<b>[ ]</b> <b>6</b> <b>[x]</b>	(6*
Random number	<b>NUM</b> <R#>	(6* R#
Add 1	<b>[+]</b> <b>1</b> <b>[ ]</b>	(6* R# + 1)
Integer	<INT>	(6* R# + 1) INT
Restore PC	<b>2nd</b> <b>[PC]</b>	(6* R# + 1) INT
Stop execution	<b>HALT</b>	R# + 1) INT HLT
Exit learn mode	<b>LEARN</b>	

(continued)

## Running the Example

Run the program. The sequence of numbers shown by your calculator will probably differ from the sample shown below.

Procedure	Press	Display
Display RUN menu	<b>RUN</b>	SELECT:
Generate number	<b>&lt;PGM&gt;</b>	3.
Generate number	<b>&lt;PGM&gt;</b>	6.
Generate number	<b>&lt;PGM&gt;</b>	1.

© 2010 Joerg Woerner  
Datamath-Calculator-Museum

Exit learn mode	<b>LEARN</b>
Stop execution	<b>HALT</b>
Restore PC	<b>2nd (PC)</b>
Integer	<b>&lt;INT&gt;</b>
Add 1	<b>1+1</b>
Random number	<b>RND (RN)</b>
Enter upper limit	<b>1 0 X</b>
Clear program	<b>2nd (OP)</b>

(continued)

Not all functions operate the same in a program as they do when executed from the keyboard. Several examples are listed in this section. Other differences between the keyboard and program operation of a function are discussed in the chapters in which the functions are defined.

---

### The Help Function

From the keyboard, **HELP** displays the prompt **SET NORMAL MODE?** and a menu from which you can choose **<YES>**, **<NO>**, or **<ESC>**. If you press **<YES>**, the calculator resets the system parameters to their default settings. If you press **<NO>**, the calculator displays all parameters that are not set to their default condition and lets you choose between resetting the parameter or leaving it unchanged. Pressing **<ESC>** clears the menu.

**HELP** as a program instruction resets the system parameters to their default settings and resets the partitions (excluding file space) to one-half data registers and one-half program memory. This is identical to pressing **HELP <YES>** from the keyboard.

### The Status Function

From the keyboard, the **LIST <ST>** (status) function lists all parameters that are not already set to their default condition and displays the error number of the most recent error.

**<ST>** as a program instruction places the number of the most recent error in the numeric display register. It does not list any status information.

### The QAD and CUB Functions

From the keyboard, the **<QAD>** and **<CUB>** selections of the **FUNC** menu prompt you to enter coefficients and display a message to identify the roots as real or complex.

In a program, no prompts or messages appear. Instead, the **<QAD>** and **<CUB>** functions get the coefficients from data registers 0, 1, 2, and 3. The results are stored in the same registers. For details, refer to Chapter 2 of the *TI-95 User's Guide*.

This section explains how to interrupt a program so that you can enter numbers into the program.

## Using **BREAK** to Enter Numbers

Many problems require you to enter a number after a sequence of keystrokes has already been performed. To solve this type of problem in a program, you need an instruction that:

- ▶ Interrupts the program at the point where you want to enter the number.
- ▶ Permits you to perform manual calculator operations such as entering a number into the display.
- ▶ Allows you to continue execution from the point in the program where it was interrupted.

The **BREAK** instruction is designed for this purpose.

**BREAK** interrupts program execution and defines **F1** as **<GO>**. By placing a **BREAK** instruction in a program, you can stop execution temporarily. When you are ready to restart the program, just press **<GO>**. If you clear **<GO>** by pressing a system-menu key before you restart the program, you can restore **<GO>** by pressing **BREAK**.

The use of a **BREAK** instruction to enter a number into a program is illustrated below.

```
RUN <PGM>
      ⋮
      BRK
Enter Number
      <GO>
      ⋮
      HLT
```

(Program Executes)

(Program Executes)

**Example** Write a program to solve the following problem.

$$Z = \sin X - \tan Y$$

This problem requires you to enter values for X and Y. Design the program so that you enter X before you run the program and Y after execution starts.

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> <1st>	
Clear program	<b>2nd</b> [CP]	
Calculate sine	<b>SIN</b>	SIN
Subtract	<b>-</b>	SIN -
Stop for number	<b>BREAK</b>	SIN - BRK
Calculate tangent	<b>TAN</b>	SIN - BRK TAN
Final result	<b>=</b>	SIN - BRK TAN =
Stop execution	<b>HALT</b>	BRK TAN = HLT
Exit learn mode	<b>LEARN</b>	

(continued)

## Making a Program Wait for Numeric Input (Continued)

### Running the Example

Before running the program, set the angle units to grads by pressing **2nd** **[DRG]** until **GRAD MODE** appears in the display. Then use the program to calculate Z for the following values of X and Y.

Value of X	Value of Y
206 Grads	94 Grads
45 Grads	101 Degrees

Procedure	Press	Display
Set grad mode	<b>2nd</b> <b>[DRG]</b> *	GRAD MODE
Display RUN menu	<b>[RUN]</b>	SELECT:
Enter X	206 <PGM>	-.0941083133
Enter Y	94 <GO>	-10.67300331
Display RUN menu	<b>[RUN]</b>	SELECT:
Enter X	45 <PGM>	.6494480483
Select ANG menu	<b>[CONV]</b> <ANG>	ANGULAR
Convert to grads	101 <D-G>	Grd = 112.2222222
Redisplay <GO>	<b>[BREAK]</b>	Grd = 112.2222222
Restart program	<GO>	5.794002064

\*Repeat until **GRAD MODE** appears in the display.

**Note:** Running a program does not clear calculations in progress. Therefore, if you decide to rerun the program after it has been stopped by the BRK instruction, press **[CLEAR]** first to clear the pending calculation.

## Pausing a Program to Display Results

This section explains how to use the pause instruction to display numeric information during program execution. For details on using pause to display messages, refer to the next chapter of this guide.

### Using Pause Instructions

A **2nd** [PAUSE] instruction stops program execution for one second to display a numeric result (or message). A pause instruction does not affect pending numeric operations, so you can use it to display intermediate results as well as final calculations.

If you want a program to pause for more than one second, include additional pause instructions in the program. For example, placing three pause instructions together in a program creates a three-second pause.

### Example

Write a program to evaluate the following equation.

$$A = B^2 * 5$$

Design the program to display the value of  $B^2$  for one second before the final result is calculated.

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> <1st>	
Clear program	<b>2nd</b> [CP]	
Square value	<b>x<sup>2</sup></b>	$x^2$
Pause	<b>2nd</b> [PAUSE]	$x^2$ PAU
Multiply by 5	<b>x</b> 5	$x^2$ PAU *5
Final result	<b>=</b>	$x^2$ PAU *5=
Stop execution	<b>HALT</b>	$x^2$ PAU *5= HLT
Exit learn mode	<b>LEARN</b>	

(continued)

**Running the Example** Run the program using  $B = 55$  and  $B = -456$ .

Procedure	Press	Display
Display <b>RUN</b> menu	<b>RUN</b>	SELECT:
Enter B	55 <PGM>	3025.
		15125.
Enter B	456 <b>+/-</b> <PGM>	207936.
		1039680.

© 2010 Joerg Woerner  
Datamath Calculator Museum

Procedure	Press	Display
Activate learn mode	<b>LEARN</b> (1st)	
Clear program	<b>2nd</b> [CPL]	
Square value	<b>x<sup>2</sup></b>	55 <sup>2</sup>
Pause	<b>2nd</b> [PAUSE]	55 <sup>2</sup> PAU
Multiply by 5	<b>x 5</b>	55 <sup>2</sup> PAU * 5
Final result	<b>=</b>	55 <sup>2</sup> PAU * 5 =
Stop execution	<b>HALT</b>	55 <sup>2</sup> PAU * 5 = HLT
Exit learn mode	<b>LEARN</b>	

(continued)

## Reference Section

---

Use this section as a source of reference information about entering calculator functions into a program, interrupting a program to enter numbers, and pausing a program to display numeric values.

---

### Restore PC

**[2nd] [PC]**—Alternately displays and erases the current value of the program counter in the second line of the display. This function operates only in the learn mode. (Use this function to restore the program counter if it has been cleared by using a system-menu key.)

### Break

**[BREAK]**—Interrupts program execution and defines the **[F1]** key as <GO>. The address where the program was interrupted is stored internally. To resume execution from that location, press <GO>. If the <GO> definition is cleared by subsequent manual calculator operations, it can be restored by pressing **[BREAK]**.

You can use **[BREAK]** as a keyboard command to stop a running program or a listing function. If you press **[BREAK]** to stop a program, you can resume execution from the interrupted location by pressing <GO>.

The instruction mnemonic for **[BREAK]** is BRK.

### Using **[BREAK]** and **[HALT]**

Although **[BREAK]** and **[HALT]** both stop program execution, the functions have differences that make them suitable for different applications:

- ▶ **[BREAK]** should be used when you want to stop a program to perform a manual calculator operation (such as entering a number) and then resume execution from the interrupted location.
- ▶ **[HALT]** should be used at the end of a program and in cases when you want to stop a program without affecting the definitions of the function keys.

### Pause

**[2nd] [PAUSE]**—Stops program execution for one second. The information displayed during the one-second pause depends upon the instructions that have been executed previously.

The instruction mnemonic for **[2nd] [PAUSE]** is PAU.



## Chapter 3: Displaying Messages

---

Although messages are most often used within a program, you can learn how to display messages by working directly from the keyboard. After you become familiar with the functions available in the alpha mode, you can use messages in your programs.

---

Table of Contents	Introduction .....	3-2
	The Alpha Menu .....	3-4
	Creating an Alpha Message .....	3-5
	Storing and Recalling Alpha Messages .....	3-8
	Editing an Alpha Message .....	3-10
	Merging Numbers with a Displayed Message .....	3-12
	Using Alpha Messages in a Program .....	3-14
	Reference Section .....	3-17

© 2010 Jerry Woelner

Diamond Calculator Museum

The alpha mode lets you display characters ranging from uppercase and lowercase letters to special signs and symbols. The exercises in this chapter demonstrate how to use the alpha mode.

### What Is the Alpha Mode?

The alpha mode causes the calculator to interpret most keystrokes as alphabetic and punctuation characters rather than as calculator functions. The scientific-calculator functions are not accessible while the calculator is in the alpha mode.

The primary keys you use in the alpha mode are arranged in the same pattern as those on a standard typewriter.

### Uses of the Alpha Mode

You can use the alpha mode anytime you want your program to display a message.

For example, you can display a message that:

- ▶ Shows the title of a program.
- ▶ Gives operating instructions as the program runs.
- ▶ Prompts you for keyboard entries.
- ▶ Identifies the result of a calculation.

If you have a printer connected to the calculator, you can print alpha messages.

### Redisplaying an Alpha Message

Messages that you create using the alpha mode are cleared by subsequent numeric operations. If you want to redisplay the last alpha message, press **OLD**. (**OLD** also restores the last user-defined function key menu, if any, as described in Chapter 4 of this guide.)

## The Alpha Mode Keyboard

When the calculator is in the alpha mode, most keys produce an alpha character. The keys have primary and second characters.

On the left side of the keyboard, the alpha character for each key is printed in blue above the key. For these keys, pressing **2nd** before pressing the key changes the character from uppercase to lowercase, or vice versa, depending upon the lowercase lock status of the keyboard.

For two of these keys, the primary alpha character is printed above and to the left of the key, and the second alpha character is printed above and to the right of the key.

- ▶ The primary alpha character for the **FUNC** key is a comma. Pressing **2nd** **FUNC** produces “.
- ▶ The primary alpha character for the **LIST** key is a period. Pressing **2nd** **LIST** produces ‘.

For most of the keys on the right side of the keyboard, the character printed on the face of a key is the key's primary alpha character. The second alpha character for each key is printed in gray above the key.

Two keys have primary alpha characters that are not printed on the keyboard.

- ▶ The primary character for the **EE** key is “.
- ▶ The primary character for the **+/-** key is |.

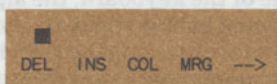
# The Alpha Menu

---

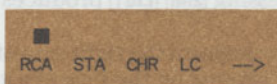
You use the **ALPHA** key to both activate and exit the alpha mode. The alpha menu has eight functions. These functions are shown below and are discussed in more detail later in this chapter.

---

**The Alpha Menu** When you press **ALPHA** to activate the alpha mode, the calculator displays the **ALPHA** indicator, a flashing cursor, and the alpha menu. If the display contained a user alpha message, the message is also displayed.



- <DEL> Deletes the character at the cursor position.
- <INS> Allows insertion of characters at the cursor position.
- <COL> Moves the cursor to a column you specify.
- <MRG> Merges a number with an alpha message.
- <-->> Displays additional selections (shown below).



- <RCA> Recalls a previous alpha message.
- <STA> Stores a message in data registers.
- <CHR> Stores a specified character code at the cursor position.
- <LC> Toggles (alternates) between lowercase lock on and off.
- <-->> Displays the previous selections (shown above).

Although the alpha register can hold 80 characters, only 16 characters can be displayed at one time. If you enter more than 16 characters, the message scrolls left to accommodate the newly entered characters. An arrow indicator appears when any characters have scrolled off the display to the left.

### Entering a Message

Before entering an alpha message, you must first activate the alpha mode by pressing **[ALPHA]**. After the alpha mode is active, you can type your message.

### Example

The following example uses the alpha mode to display the message **HELLO THERE**.

Procedure	Press	Display
Clear calculator	<b>[CLEAR]</b>	0.
Activate alpha mode	<b>[ALPHA]</b>	■
Begin alpha message	HELLO	HELLO■
Enter "space"	<b>[SPACE]</b>	HELLO ■
Continue message	THERE	HELLO THERE■
Exit alpha mode	<b>[ALPHA]</b>	HELLO THERE

When you exit the alpha mode, any message longer than 16 characters scrolls across the display and stops when the last 16 characters are visible.

- ▶ You can press **[←]** to see the first part of the message again.
- ▶ You can press **[→]** to skip to the end of the message instead of waiting for it to scroll.

(continued)

## Using Upper and Lower Case

The first time you activate the alpha mode after turning the calculator on, the keyboard is locked in uppercase letters. You can toggle between uppercase and lowercase lock by pressing <LC> from the alpha menu. The LC indicator shows when lowercase lock is in effect. The case that you select remains in effect until you change it or turn the calculator off.

- To enter a lowercase letter while uppercase lock is in effect, press **2nd** before pressing the key for the letter.
- To enter an uppercase letter while lowercase lock is in effect, press **2nd** before pressing the key for the letter.

**Note:** If you press the **2nd** key by mistake, pressing it again cancels the shift request.

## Example

The following example uses both uppercase and lowercase characters to display the word **Radius**.

Procedure	Press	Display
Clear calculator	<b>CLEAR</b>	0.
Activate alpha mode	<b>ALPHA</b>	■
Enter first letter in uppercase	R	R■
Select lowercase lock	<--> <LC>	R■
Complete the word in lowercase	ADIUS	Radius■
Reselect uppercase lock	<LC>	Radius■
Exit alpha mode	<b>ALPHA</b>	Radius

**Using the <CHR> Selection** The <CHR> selection on the alpha menu lets you specify a character by referring to its character code. (For a listing of these codes, see Appendix C.) This function is particularly useful when you want to:

- ▶ Include printer control codes (such as a carriage return) in an alpha message.
- ▶ Display special characters (such as  $\Sigma$ ,  $\bar{X}$ , and  $+$ ) in an alpha message.

To use the <CHR> selection:

1. Activate the alpha mode and position the cursor where you want the character to appear.
2. Select <CHR> from the alpha menu.

3. Enter the three-digit code for the character. The character is displayed at the cursor position.

If the character is a control code for the printer, exit the alpha mode and press **2nd** [PRINT] to send the code to the printer.

### Example

This example uses <CHR> to send a carriage-return command to the printer.

Procedure	Press	Display
Clear calculator	<b>CLEAR</b>	0.
Activate alpha mode	<b>ALPHA</b>	■
Enter character code	<--> <CHR> 013 \	■
Exit alpha mode	<b>ALPHA</b>	\
Send code to printer	<b>2nd</b> [PRINT]	\

## Storing and Recalling Alpha Messages

You can easily redisplay an alpha message that has been erased by a numeric value. However, to reuse a message that has been replaced by another alpha message, you must either reenter the original message or store the original message in data registers when you create it.

### How Alpha Messages Occupy Memory

When you store an alpha message, it is stored in data registers, just as numeric values are. To avoid a conflict with previously stored alpha or numeric data, you need to understand how alpha messages occupy memory.

A stored alpha message occupies 10 data registers. This is because the entire contents of the 80-character alpha register are stored. If the message is fewer than 80 characters in length, the remaining portion of the alpha register contains blanks.

For example, if you store the message "RESULT =" in data register 000, it occupies data registers 000 through 009.

Register	Contents
000	R E S U L T =
001	
.	(registers 002 through 007)
008	
009	

Although data registers 001 through 009 are filled with blanks, they are part of the alpha message and should not be used to store any other data.

Before designating a location to store a message, make sure you have 10 consecutive data registers available that do not contain important information.

### Storing an Alpha Message

The <STA> (store alpha) selection lets you store the currently displayed alpha message in a specified series of data registers. This function is primarily useful when you plan to redisplay an alpha message after displaying another alpha message.


To store an alpha message in data registers:

1. Activate the alpha mode and create the message.
2. Press <STA> and enter the address of the first of the 10 data registers you want to hold the message.

### Recalling an Alpha Message

The <RCA> (recall alpha) selection lets you redisplay either the most recently displayed message or a message you have stored using <STA>.

To redisplay an alpha message:

1. Activate the alpha mode.
2. Use the key sequence that applies to you.
  - To redisplay the most recently displayed message after it has been erased by a numeric value, press <RCA> . The message appears with the cursor positioned where it was before the number was displayed.
  - To redisplay a message you have stored using <STA>, press <RCA> and enter the data register where you stored the message. The message appears with the cursor positioned immediately following the last nonblank character.

**Note:** If you use <RCA> to recall the contents of data registers that do not contain a stored alpha message, the calculator displays the alpha characters that correspond to the numeric values stored in the registers.

## Editing an Alpha Message

---



You can use the editing features of the alpha mode to replace existing characters, insert additional characters, or delete unwanted characters in a displayed alpha message. If the message is not currently displayed, you must first redisplay it using the <RCA> selection described on the previous page.

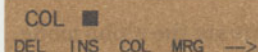
---

### Positioning the Cursor

Before editing an alpha message, you must position the cursor at the place in the message where you want to make a change.

Use the following keys to position the cursor.

-  Moves the cursor to the left, one position at a time.
-  Moves the cursor to the right, one position at a time.
- <COL> Lets you move the cursor to a specified character position (column) in the alpha register by prompting you to enter a column number. Columns are numbered 1 through 80, starting with the leftmost column.



COL █  
DEL INS COL MRG -->

After you enter the column number, the alpha message is redisplayed with the cursor in the specified column.

After positioning the cursor, you can use the editing features to change the message.

### Replacing Characters

To replace a character in an alpha message:

1. Position the cursor on the character you want to replace.
2. Type the new character.

## Inserting Characters

To insert additional characters in an alpha message:

1. Position the cursor on the character before which you want to insert characters.
2. Press **<INS>**.

The **INS** indicator shows that the insert mode is active.

3. Enter the characters you want to insert.

Any of the following actions cancel the insert mode.

- ▶ Moving the cursor (using **←**, **→**, or **<COL>**).
- ▶ Selecting **<DEL>**, **<RCA>**, **<STA>**, **<MRG>**, or **<-->>**.
- ▶ Pressing **CE** or **CLEAR**.
- ▶ Exiting the alpha mode.

## Deleting Characters

To delete unwanted characters from an alpha message:

1. Position the cursor over the first character to be deleted.
2. Press **<DEL>** once for each character you want to delete.

## Clearing an Alpha Message

To clear an alpha message, press **CE** or **CLEAR** while the calculator is in the alpha mode.

## Merging Numbers with a Displayed Message

---

Merging lets you display a numeric value together with an alpha message. This feature is particularly useful when you need to display an explanatory message with the result of a calculation. The numeric value can be either the value in the numeric display register or a value stored in a data register.

---

### How to Specify the Merge Point

Specifying a merge point takes some planning. The rightmost digit (or the decimal point if the number is an integer) of the number will occupy the current cursor position. Because a number may require as many as 12 character positions (16 for unformatted numbers), you must leave enough space for the number.


You may find it helpful to draw a template of the character positions in the display to help you establish an appropriate merge point.

For example, before merging a number with the message "AREA =", you may wish to position the cursor at column 16.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	R	E	A	=											■

### Using the Merge Function

To merge a number with an alpha message, follow these steps.

1. Activate the alpha mode and create (or recall) the alpha message.
2. Position the cursor where you want the rightmost digit of the merged number to appear.
3. To merge the number in the numeric display register, press <MRG> .

To merge a stored number, press <MRG> and enter the data register where the number is stored.

**Note:** The number is displayed in the currently selected display format (such as FIX, EE, ENG, or HEX).

Your program can display messages and execute most of the functions available through the alpha menu.

**Example** The following example merges a stored number with an alpha message. Before beginning this example, make sure the calculator is not in the alpha mode.

Procedure	Press	Display
Store 64 in register A	64 <b>[STO]</b> A	64.
Activate alpha mode	<b>[ALPHA]</b>	■
Enter the message	AREA <b>[=]</b>	AREA = ■
Allow room for the number	<COL> 16	AREA = ■
Merge the number	<MRG> A	REA = 64.■
Exit alpha mode	<b>[ALPHA]</b>	AREA = 64.

### Datamath Calculator Museum

**Note:** Because 64 is also in the numeric display register in this case, you could optionally use <MRG> **[=]**.

## Using Alpha Messages in a Program

---

Your program can display messages and execute most of the functions available through the alpha menu.

---

### Replacing or Appending a Message

When you make an alpha entry directly from the keyboard, you can easily see whether the entry starts a new message or is appended to an existing message.

However, when you make an alpha entry while entering program instructions, you must visualize what the calculator **would** display if program execution were interrupted just prior to your alpha entry (for example, by a halt or pause instruction).

- ▶ If a numeric value would be displayed, any new alpha entry starts a new alpha message.
- ▶ If a user alpha message would be displayed, any new alpha entry is appended to the existing message, starting at the cursor position in the existing message.

In most cases, you can easily visualize whether the most recent instructions leave a numeric value or a user message in the display. However, you should be aware that some instructions, when executed, do not affect the display. These include such instructions as CMS, PRT, ADV, and the transfer instructions discussed in the next chapter.

### Avoiding Problems

If you are uncertain about the effect of starting an alpha entry at a specific point in your program, use these guidelines.

- ▶ When you want to be sure a new entry starts a new message, precede the entry with a CE instruction.
- ▶ When you want to be sure a new entry is appended to the most recent alpha message, precede the new entry with an RCA = instruction.

**Additional Differences**

As you enter an alpha message into a program, the message is displayed in single quotes to help you distinguish alpha messages from instruction mnemonics.

The <LC> (Lowercase toggle) function cannot be stored as a program instruction, but you can use the function to toggle lowercase lock on or off while the calculator is in the learn mode.

If you want your program to reposition the cursor (for example, to prepare for merging a number), you must use the <COL> function as a program instruction. You cannot store the  $\left[ \leftarrow \right]$  or  $\left[ \rightarrow \right]$  keys as program instructions because these keys are used in displaying the program.

(continued)

**Example Program** This example stores a program that creates two alpha messages: one that prompts you to enter the radius of a sphere, and one that labels the volume of the sphere.

Procedure	Press	Display
Enter learn mode	<b>LEARN</b> <1st>	
Clear old program	<b>2nd</b> [CP]	
Activate alpha mode	<b>ALPHA</b>	
Enter first message	ENTER RADIUS	'ENTER RADIUS'
Exit alpha mode	<b>ALPHA</b>	'ENTER RADIUS'
Wait for entry	<b>BREAK</b>	ER RADIUS' BRK
Calculate volume	$y^x$ 3 x <b>2nd</b> [ $\pi$ ] x 4 + 3 =	$y^x 3^* \pi * 4/3 =$
Activate alpha mode	<b>ALPHA</b>	$y^x 3^* \pi * 4/3 =$
Enter second message	VOL [=]	I *4/3= 'VOL ='
Position cursor and merge volume	<COL> 16 <MRG> [=]	' COL 16 MRG =
Exit alpha mode	<b>ALPHA</b>	' COL 16 MRG =
Mark end of program	<b>HALT</b>	L 16 MRG = HLT
Exit learn mode	<b>LEARN</b>	

## Running the Program

To run the program, press **RUN** <PGM>. When prompted, enter a value for the radius and press <GO>. The program displays the labeled result.

Use this section as a source of reference information for using the alpha mode.

### Alpha Mode

**[ALPHA]**—Activates or exits the alpha mode, depending on the state of the calculator. Pressing **[ALPHA]** when the alpha mode is not active displays the alpha mode menu and the **ALPHA** indicator. Any user alpha message in the display remains in the display when you activate the alpha mode. The cursor is positioned in the same column as the last time the alpha mode was activated. Pressing **[ALPHA]** when the calculator is already in the alpha mode exits the alpha mode and turns off the **ALPHA** indicator.

If the learn mode is active when you select the alpha mode, the characters and most alpha mode functions that you enter are stored in program memory. In this case, exiting the alpha mode returns to the learn mode. The **[ALPHA]** key is not stored as a program instruction. In a program, alpha messages are displayed enclosed in single quotes.

### Alpha Register

The calculator retains the most recent alpha message in an 80-character alpha register. When the calculator is first turned on, the default message stored in this register is **TI-95 PROCALC**. When the calculator is in the alpha mode, a message in this register can be redisplayed with the **<RCA> [=]** function. Outside of the alpha mode, you can redisplay the last alpha message by pressing **[OLD]**. (**[OLD]** also restores the last user-specified definitions for the function keys **F1–F5**.)

If the message in the alpha register is not currently displayed when a new alpha entry is made, the register is cleared and the entry starts a new alpha message.

If the message in the alpha register is displayed when a new alpha entry is made, the new entry is appended to the existing message, starting at the cursor position in the existing message.

(continued)

**← and →**

If the alpha mode is active, ← and → move the alpha cursor one position to the left or right.

If the alpha mode is not active, ← restarts the scrolling of any long (more than 16 characters) alpha message currently in the display. → bypasses any scrolling in progress by displaying the last 16 characters of the message.

← and → can be used only as keyboard commands. You cannot store either key as a program instruction.

**Delete Character**

**ALPHA** <DEL>—Deletes the character at the current position of the cursor in a displayed alpha message. Any characters to the right of the cursor move left to fill the vacated position. If there are no characters to the right of the cursor, <DEL> places a space at the cursor position. <DEL> can be used as a keyboard command or as a program instruction.

**Insert Mode**

**ALPHA** <INS>—Allows insertion of characters in a displayed alpha message. An inserted character occupies the position of the cursor. Remaining characters, including the character previously located at the cursor position, are moved to the right. If the alpha display register already contains 80 characters, inserting a character causes the last character in the message to be discarded.

<INS> can be used as a keyboard command or as a program instruction.

When you use <INS> as a keyboard command, the **INS** indicator appears, indicating you are inserting characters. Any keystroke that does not produce an alpha character cancels the alpha insert mode and turns off the **INS** indicator. (In a program, you can access the **CHR** function without cancelling the insert mode.)

**Move Cursor  
to Column**

**[ALPHA]** <COL> *nn*—Positions the alpha cursor at column *nn* in the displayed alpha message. The value *nn* must be in the range 01 to 80. <COL> can be used as a keyboard command or as a program instruction.

**Merge Number  
into Alpha  
Message**

**[ALPHA]** <MRG> *nnn* or *X*—Merges the numeric value stored in data register *nnn* or *X* with the displayed alpha message. **[ALPHA]** <MRG> **[=]** merges the number from the numeric display register with the displayed alpha message.

The number is merged to the left of the cursor, with the last digit (or the decimal point if the number is an integer) at the current cursor position. The cursor is then advanced to the next position. If characters already exist in the positions required by the number, the characters are replaced by the corresponding digits of the number.

**Note:** If there are not enough character positions to the left of the cursor for the merged number, the affected digits of the number are not merged.

The format of the merged number is determined by the current display format. A number may require as many as 12 character positions (16 for unformatted display mode).

Either form of <MRG> can be used as a keyboard command or as a program instruction.

**Recall Last  
Alpha Message**

**[ALPHA]** <RCA> **[=]**—Displays the alpha message currently contained in the alpha register. If the existing message is not currently displayed, <RCA> **[=]** must precede any new alpha entry that is to be appended to the message. The cursor is positioned in the same column as when the message was last displayed. <RCA> **[=]** can be used as a keyboard command or as a program instruction.

(continued)

### Recall Stored Alpha Message

**[ALPHA]** <RCA> *nnn* or *X*—Displays the stored alpha message that begins in register *nnn* or *X*. The message replaces any message currently contained in the alpha register. The cursor is positioned immediately following the last nonblank character in the message. <RCA> *nnn* or *X* can be used as a keyboard command or as a program instruction.

### Store Alpha Message

**[ALPHA]** <STA> *nnn* or *X*—Stores the alpha message currently contained in the alpha register starting at data register *nnn* or *X*. The stored message occupies 10 consecutive data registers. <STA> can be used as a keyboard command or as a program instruction.

### Display Specified Character

**[ALPHA]** <CHR> *nnn*—Stores the character corresponding to the number *nnn* in the alpha register at the current cursor position and advances the cursor to the next position. The number *nnn* must be in the range 000 through 255 decimal. (For a table of the characters, refer to Appendix C.) <CHR> can be used as a keyboard command or as a program instruction.

### Toggle Lowercase Lock

**[ALPHA]** <LC>—Turns lowercase lock on or off depending on the current state of the calculator. If lowercase lock is off (the default state), <LC> turns it on and displays the LC indicator. If lowercase lock is on, <LC> turns it off and turns off the LC indicator.

The state of lowercase lock affects the case of letters you enter in an alpha message or an alpha field. When lowercase lock is on, pressing **[2nd]** causes the next letter entered to be uppercase. When lowercase lock is off, pressing **[2nd]** causes the next letter entered to be lowercase.

You can use <LC> in the learn mode for entering uppercase or lowercase letters, but you cannot store <LC> as a program instruction.

## Chapter 4: Controlling the Sequence of Operations

---

The previous chapters showed you how to write programs that sequentially execute a series of steps. This chapter describes instructions that enable you to change the order in which program steps are executed.

---

<b>Table of Contents</b>	<b>Introduction</b> . . . . .	<b>4-2</b>
	<b>Before Proceeding with this Chapter</b> . . . . .	<b>4-4</b>
	<b>Using Program Labels</b> . . . . .	<b>4-5</b>
	<b>Using Go To Label</b> . . . . .	<b>4-8</b>
	<b>Using Go To</b> . . . . .	<b>4-10</b>
	<b>Using Subroutines in a Program</b> . . . . .	<b>4-11</b>
	<b>Programming the Function Keys</b> . . . . .	<b>4-16</b>
	<b>Creating a Function-Key Menu</b> . . . . .	<b>4-18</b>
	<b>Restoring a User-Defined Menu</b> . . . . .	<b>4-21</b>
	<b>Clearing Function-Key Definitions</b> . . . . .	<b>4-22</b>
	<b>Listing Program Labels</b> . . . . .	<b>4-24</b>
	<b>Speeding Up Program Execution</b> . . . . .	<b>4-25</b>
	<b>Reference Section</b> . . . . .	<b>4-26</b>

©2010 David Wehner  
Desamain Calculator Museum

The calculator normally executes program steps in the same order in which you stored them. For problems that are relatively simple, you can write a “straight-line” program that executes a sequence of keystrokes once, from beginning to end. Many problems, however, cannot be solved efficiently by a straight-line program.

---

### Why Change the Sequence?

You may have a problem, for example, that requires the same key sequence to be executed many times. Although you could probably repeat the key sequence every place it is needed, the program could become very large, perhaps exceeding the memory capacity of the calculator. You could also spend a lot of time storing the program in memory.

Transfer instructions allow you to solve problems involving repetition by directing the calculator to execute a key sequence as many times as needed. By using these instructions, you can shorten a program and make it easier to enter and edit.

Besides enabling a program to repeat a key sequence, transfer instructions allow a program to skip a sequence of keystrokes. This feature is particularly powerful when transfer instructions are used in conjunction with decision-making instructions, as described in the next chapter.

**The Transfer Functions** The transfer functions are listed below.

Mnemonic	Action
GTL	In a program, GTL (go to label) transfers control to a program step you have labeled. From the keyboard, GTL sets the program counter to a program step you have labeled, but does not start program execution.
GTO	In a program, GTO (go to) transfers control to the specified program step. From the keyboard, GTO sets the program counter to a specified program step, but does not start program execution.
SBL	In a program or from the keyboard, SBL (subroutine label) transfers control to a subroutine you have labeled.
SBR	In a program or from the keyboard, SBR (subroutine) transfers control to the subroutine at the specified program step.

**Note:** Within a program, RUN can be used with a transfer instruction to transfer control between programs that are stored in separate areas. For information on this use of transfer instructions, refer to page 8-34.

#### The DFN Instruction

The DFN (define) instruction lets you create your own function-key menus. These menus, referred to as **user-defined** menus, control the order of program execution based upon the function key that you press.

The DFN instruction can be used only within a program.

## Before Proceeding with this Chapter

Before working the examples in the remainder of this guide, you should be familiar with the basic programming skills covered in the first three chapters. If you have difficulty with an example, refer to those chapters for instructions.

### Assumptions

From this point forward, you should know how to activate and exit the learn mode, clear program memory, display the program counter, enter uppercase and lowercase messages, and run a program stored in program memory.

### Format for Program Examples

Up to now, program examples have been presented in a format that shows each keystroke required to enter a program. In remaining chapters:

- ▶ Examples show only the mnemonic form of a program instruction. For example, PAU represents the key sequence [2nd] [PAUSE]. If you don't recognize a mnemonic, refer to Appendix C for a complete list of instruction mnemonics.
- ▶ Alpha messages are shown in single quotes. You must remember to activate and exit the alpha mode to enter these messages. Alpha key sequences, where shown, assume that your keyboard is not in lowercase lock (LC indicator not visible in the display).
- ▶ Functionally related instructions are grouped together. The instructions are not necessarily grouped as they would be in a printed listing.

For example, the last program in Chapter 3 is shown below in mnemonic form.

PC =	Program Mnemonics	Comments
0000	'ENTER RADIUS'	Creates message
0012	BRK	Waits for radius
0013	$y \times 3 \times \pi$	Calculates volume
0022	'VOL ='	Creates message
0026	COL 16 MRG =	Positions cursor and merges
0030	HLT	Stops program

A program label is an instruction that you can use to identify a particular location in a program. Typically, you use a label to mark the beginning of a sequence of instructions to which you plan to transfer program control.

### Labeling a Program Segment

To place a label instruction in a program, use the key sequence:

**[2nd] [LBL] aa**

where *aa* is any two alphanumeric characters. For example, **[2nd] [LBL] AA** places the mnemonic "LBL AA" in a program at the current location of the program counter.

After you press **[2nd] [LBL]**, the calculator interprets your next keystrokes as alphanumeric characters. This eliminates the need for you to activate the alpha mode to enter the two characters of the label. You can use any combination of uppercase and lowercase letters, digits, and punctuation symbols in the label name.

Some examples of labels are shown below.

Label	Key Sequence
LBL fx	<b>[2nd] [LBL] [2nd] F [2nd] X</b>
LBL 10	<b>[2nd] [LBL] 10</b>
LBL Z1	<b>[2nd] [LBL] Z1</b>
LBL Aa	<b>[2nd] [LBL] A [2nd] A</b>

You can label any part of a program; the presence of the label does not interfere with program execution or any calculations in progress in the program.

You can use as many labels as you need in a program. However, you should not use the same label more than once in the same program. If you repeat a label, any transfer to that label is always directed to the first occurrence of the label. (The calculator begins searching for a label at program address 0000 and will not find the additional occurrences of the label.) Refer to "Listing Program Labels" in this chapter for details on listing the labels used in program memory.

(continued)

### Why Use Labels?

The field of a transfer instruction must specify a transfer location. Although you can identify the transfer location by giving its step address, it is more flexible to use a label. A label provides you with a method of identifying a program location that is not dependent upon the numeric address of the location.

The step address of an instruction changes when you insert or delete instructions ahead of it. By using a label to identify a program location, you do not have to keep track of these changes. The relative location of the label remains constant, eliminating the need to correct any numeric transfer addresses each time you insert or delete other program instructions.

Consider a program that contains the segment shown below. If you insert a RCL B instruction before this segment, you change the address of all instructions that follow the inserted instruction.

Before Insertion		After Insertion	
PC =	Mnemonics	PC =	Mnemonics
0130	* PI =	0130	RCL B
0133	PAU	0132	* PI =
		0135	PAU

Without labels, you must change any transfer references to the \* instruction originally located at 0130 to show that the new location of the instruction is 0132. If a program has many such references, correcting them is time-consuming and can cause mistakes.

The GTL (go to label) instruction transfers program control to the first instruction following a specified label.

### Why Use Labels? (Continued)

With labels, you can insert an instruction without correcting any references.

#### Before Insertion

PC = Mnemonics

0130 LBL AA  
0133 \* PI =  
0136 PAU

#### After Insertion

PC = Mnemonics

0130 RCL B  
0132 LBL AA  
0135 \* PI =  
0138 PAU

The segment still begins at the point marked by LBL AA. Any transfer to LBL AA now transfers to step 135 instead of step 133.



## Using Go To Label

The GTL (go to label) instruction transfers program control to the first instruction following a specified label.

### Transferring Control to a Label

The normal order of program execution is from program step 0000 to the end of the program. By including a GTL instruction in the program, you can transfer program control to any labeled location in the program. The program runs sequentially from the new location until another transfer instruction is executed or the program is stopped.

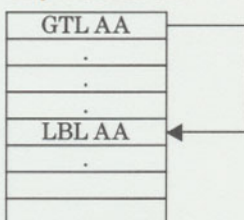
To enter the GTL instruction, use the key sequence:

**2nd** [GTL] *aa*

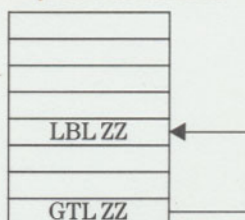
where *aa* corresponds to the label in the program.

The following illustrations show how the GTL instruction affects the flow of program execution. In the illustration on the left, the GTL instruction causes the program to skip several instructions. In the illustration on the right, the GTL instruction causes the program to repeat several instructions. Program sequences such as that on the right are called **loops**.

#### Skip Instructions



#### Repeat Instructions



**Example**

A counting loop is a good way to illustrate a transfer instruction. The following program uses a transfer instruction to repeat the key sequence  $+ 2 = \text{PAU}$ . The PAU instruction is included in the program so that you can see the result of the  $+ 2 =$  operation. The GTL AA instruction transfers control back to the beginning of the program, causing the sequence to be repeated endlessly.

By repeating the sequence, the GTL instruction creates a loop that counts by twos.

PC =	Program Mnemonics	Comments
0000	LBL AA	Labels segment
0003	$+ 2 =$	Adds 2
0006	PAU	Pauses for one second
0007	GTL AA	Transfers control to label AA

Because the program does not include a means to exit the loop, this type of loop is called an **infinite loop**. To stop an infinite loop, you must press **HALT** or **BREAK**. Methods for exiting a loop under program control are described in the next chapter.

**Running the Example**

Run the program.

Procedure	Press	Display
Clear the display	<b>CLEAR</b>	0.
Run the program	<b>RUN</b> <PGM>	2.
		4.
Stop the program	<b>HALT</b>	6.

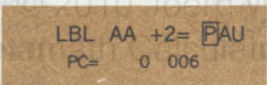
In some cases, you may prefer to transfer control to an instruction by referring to its program address instead of using a label. The address of an instruction is the step number of the instruction as shown by the calculator's program counter.

### Determining the Address of an Instruction

To find the program address of an instruction you have stored in program memory:

1. Press **[LEARN]** and use <1st>, <PC>, or <END> to enter the learn mode at the point nearest the instruction.
2. Use the **[→]** and **[←]** keys to position the cursor on the instruction. The address of the instruction is shown by the program counter.

For example, suppose you want to find the program address of the PAU instruction in the program entered on page 4–9. If you apply the procedure described above to place the cursor over PAU, the display should show:



LBL AA +2= PAU  
PC= 0 006

In this case, the address of the PAU instruction is 0006.

### Transferring Control by Address

The GTO (go to) instruction transfers control to a specified program address. To enter a GTO instruction in a program, use the key sequence

**[INV]** **[2nd]** **[GTL]** *nnnn*

where *nnnn* represents the address of the instruction you want to execute next. For example, GTO 0150 transfers control to the instruction at program address 0150.

You can use short-form addressing, as described in the “Memory Operations” chapter of the *TI-95 User's Guide*, to specify the address.

As you begin to write programs that are more complex, you may find that you need to execute the same sequence of instructions from several different locations in the program. Although you could store the sequence at each point where you want it to execute, you can save yourself time by designing the sequence as a subroutine and storing it only once.

## What Is a Subroutine?

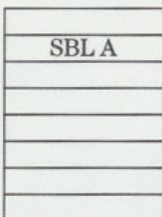
A subroutine is a sequence of instructions that can be executed, or **called**, from any point in the program. When you call a subroutine, control is transferred to the beginning of the subroutine. After the subroutine has executed, control returns to the instruction that follows the subroutine call. The word “call” is understood by programmers to mean that control will return to the original segment after the subroutine has been executed.

To include a subroutine in a program, use the following procedure.

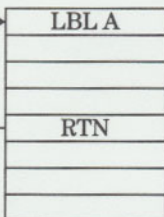
1. Store a label at the beginning of the subroutine. (Although you can transfer control to the step address of a subroutine, using a label has the advantages stated earlier.)
2. Store the instructions that make up the subroutine.
3. Store a RTN (return) instruction at the end of the subroutine. To enter a return instruction, press **2nd** [RTN].

The following illustration shows how a subroutine affects the flow of program execution. The SBL instruction (described later in this section) is used to call the subroutine.

### Main Program



### Subroutine

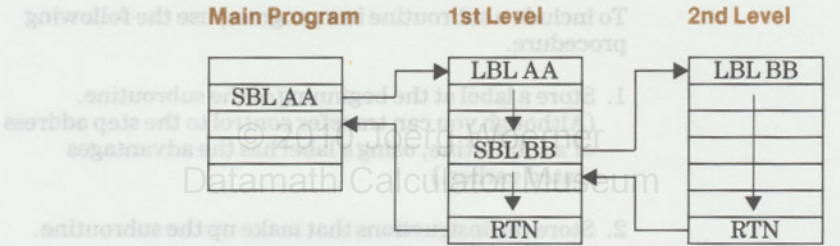


(continued)

As you begin to write programs that are more complex, you may find that you need to execute the same sequence of instructions from several different locations in the program. Although you could store the sequence at each point where you want it to execute, you can save yourself time by designing the sequence as a subroutine and storing it only once.

**Levels of Subroutines** You can design a program so that one subroutine calls another subroutine. In this case, the RTN instruction at the end of the second subroutine returns control to the first subroutine, which returns control to the original program segment.

The calculator allows a maximum of eight "levels" of subroutines to be active at one time. The illustration below shows two levels of subroutine calls.



When the calculator encounters a RTN instruction, it returns control to the program segment or subroutine that called the current level of subroutine. If the calculator encounters a RTN instruction when no levels are active, the program stops. (RTN operates like HLT if there are no subroutine levels active.)



(continued)

---

**Calling a Subroutine by Label**

The SBL (subroutine label) instruction transfers program control to a subroutine that begins with a specified label.

To enter the SBL instruction, use the key sequence

**[2nd] [SBL] aa**

where *aa* represents the label of the subroutine you want to call.

After the subroutine has executed, program control returns to the instruction following the SBL instruction.

**Calling a Subroutine by Address**

The SBR (subroutine) instruction lets you transfer program control to a subroutine by referring to the subroutine's program address.

To enter the SBR instruction, use the key sequence

**[INV] [2nd] [SBL] nnnn**

where *nnnn* represents the step address of the first instruction in the subroutine.

After the subroutine has executed, program control returns to the instruction following the SBR instruction.

(continued)

### Avoiding Difficulties in Subroutines

To avoid some common problems that can occur when using subroutines in programs, keep these suggestions in mind.

- ▶ To prevent the accidental execution of a subroutine, make sure the program segment preceding it ends with a RTN, HLT, or transfer instruction.
- ▶ If the subroutine needs an intermediate result, use parentheses instead of [=] to perform the calculation. This avoids completing calculations in progress in the program segment that called the subroutine.
- ▶ If you need to clear the display within a subroutine, use a numeric entry of 0 instead of [CLEAR]. [CLEAR] clears all calculations in progress.
- ▶ You should not use a subroutine to call itself. Using a subroutine to call itself will generally result in a SBR STACK FULL error.

(continued)

By "defining" the function key, you can create a menu that lets you transfer control to any of several locations in the program, depending on the function key you press.

**Example** This example illustrates a simple subroutine (labeled PZ) that multiplies the contents of the numeric display register by 2, adds 1, and pauses for one second before returning control to the main program. The main program calls the subroutine to perform the calculation on the numbers 2 and 9.

PC =	Program Mnemonics	Comments
0000	LBL AA	Labels segment
0003	CLR	Clears calculator
0004	2 SBL PZ	Calls subroutine
0008	9 SBL PZ	Calls subroutine again
0012	CLR	Clears calculator
0013	HLT	Stops program
0014	LBL PZ	Labels subroutine
0017	(*2+1)	Performs calculation
0023	PAU	Pauses to display result
0024	RTN	Returns from subroutine

## Running the Example

Run the program.

Procedure	Press	Display
Run the program	<b>RUN</b> <PGM>	5.
		19.
		0.

# Programming the Function Keys

---

By “defining” the function keys, you can create a menu that lets you transfer control to any of several locations in the program, depending on the function key you press.

---

## Definable Function Keys

During normal calculator operation, the function keys are defined by the system menus. For example, when you press **CONV**, the calculator defines the function keys to provide a variety of conversions.

You can create your own function-key definitions by executing a **2nd** **[DFN]** instruction within a program (DFN cannot be used as a keyboard command). The DFN (define) instruction is followed by a field that specifies:

- ▶ The function key to define.
- ▶ The three-character message to display above the key. (This lets you display information to describe the definition of the key.)
- ▶ The label of the program segment to execute when you press the function key. (The sequence of instructions following the label determines what operations are performed when you press the key.)

The three-character message above the function key is not displayed until program execution is stopped or paused.

## Storing the Instruction

To include a DFN instruction in a program:

1. Press **2nd** **[DFN]**.
2. Press the function key (**[F1]**–**[F5]**) you are defining.
3. Enter the three-character message you want displayed above the key. If your message has fewer than three characters, use spaces as blank characters.
4. Enter the label where you want to start execution when the function key is pressed.

When using a menu, you must design the program so that each function key transfers control to a specific program segment. Typically, you place a HLT instruction at the end of each segment to prevent the calculator from executing past the segment.

**Example** The following keystrokes store a DFN instruction in program memory. This instruction displays “SUB” above **[F1]** and instructs the calculator to transfer program execution to label AA when you press **[F1]**.

Procedure	Press	Display
Define <b>[F1]</b> key	<b>[2nd] [DFN]</b>	DFN
	<b>[F1]</b>	DFN F1
	SUB	DFN F1:SUB@
	AA	DFN F1:SUB@AA

The calculator automatically accepts alpha characters for the function description and label name. It also supplies the “:” after you enter the first character of the function description and the “@” after you enter the last character of the description. You can read this instruction as “Define key F1 as SUB at label AA.”

## Creating a Function-Key Menu

When using a menu, you must design the program so that each function key transfers control to a specific program segment. Typically, you place a HLT instruction at the end of each segment to prevent the calculator from executing past the segment.

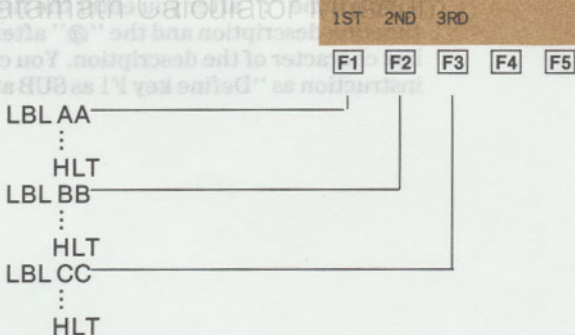
### Building a Menu

To create a menu in a program, use a separate DFN instruction for each function key you want to define. After the final function key is defined, put a HLT instruction at the point in the program where you want the definitions displayed.

The following illustration shows how you structure a program for a three-function menu. Pressing **[F1]**, **[F2]**, or **[F3]** transfers program execution to label AA, BB, or CC, respectively.

```
DFN F1:1ST@AA
DFN F2:2ND@BB
DFN F3:3RD@CC
HLT
```

These instructions define the menu shown below.



**Example  
Program 1**

Write a program that creates a menu with options to calculate the third, fourth, or fifth root of a number.

PC =	Program Mnemonics	Comments
0000	'ROOTS'	Creates menu title
0005	DFN F1:3RD@AA	Defines F1
0012	DFN F2:4TH@BB	Defines F2
0019	DFN F3:5TH@CC	Defines F3
0026	HLT	Stops program and displays menu
0027	LBL AA	Labels segment
0030	(INV $y^x$ 3)	Calculates 3rd root
0035	HLT	Stops program
0036	LBL BB	Labels segment
0039	(INV $y^x$ 4)	Calculates 4th root
0044	HLT	Stops program
0045	LBL CC	Labels segment
0048	(INV $y^x$ 5)	Calculates 5th root
0053	HLT	Stops program

**Running  
Example 1**

Test the program by calculating the third and fifth roots of a number.

Procedure	Press	Display
Activate the menu	<b>RUN</b> <PGM>	ROOTS 3RD 4TH 5TH
Enter a number	8	8
Calculate 3rd root	<3RD>	2.
Enter a number	3125	3125
Calculate 5th root	<5TH>	5.

(continued)

**Example Program 2** This example lets you use the function keys to enter sides a and b of a right triangle. When you press <CAL>, the program calculates the length of the hypotenuse. (If the keyboard is not in lowercase lock, use **2nd** A and **2nd** B to enter the lowercase letters a and b.)

PC =	Program Mnemonics	Comments
0000	'ENTER SIDES'	Creates menu title
0011	DFN F1:a @SA	Defines F1
0018	DFN F2:b @SB	Defines F2
0025	DFN F5:CAL@CH	Defines F5
0032	HLT	Stops program and displays menu
0033	LBL SA	Labels segment
0036	STO A HLT	Stores side a in reg. A
0039	LBL SB	Labels segment
0042	STO B HLT	Stores side b in reg. B
0045	LBL CH	Labels segment
0048	(RCL A x^2	Calculates hypotenuse
0052	+ RCL B x^2) SQR	
0058	'HYP='	Creates alpha message
0062	COL 16 MRG =	Merges result
0066	HLT	Stops program

**Running Example 2** Test the program by entering values that describe a "3-4-5" triangle.

Procedure	Press	Display
Activate the menu	<b>RUN</b> <PGM>	ENTER SIDES a b CAL
Enter side a	30 <a>	30.
Enter side b	40 <b>	40.
Calculate hypotenuse	<CAL>	HYP= 50.

If you use a system menu while a menu of yours is displayed, the system menu will replace your menu. You can restore your menu definitions by pressing **OLD**.

---

### Restoring Your Menus

If you clear your menu by using a system-menu key or pressing **2nd** **[F:CLR]**, you can restore the menu and any previous alpha message by pressing **OLD**. You can also restore the menu by executing **OLD** as a program instruction.

You cannot use **OLD** to restore a system menu.

For example, suppose your program displays a menu that lets you enter several variables. You want to perform a metric conversion before entering one of the variables. When you press **CONV**, your menu is replaced by the **CONVERSIONS** menu. After you make the conversion, press **OLD** to restore your menu.

Once a menu has been defined by a program, that menu can be restored by **OLD** until you:

- ▶ Replace the menu with another user-defined menu.
- ▶ Clear the menu with a clear function-key instruction, as explained in the next section.
- ▶ Turn off the calculator.
- ▶ Run another program.
- ▶ Run the same program again, but stop execution prior to defining the menu.

To restore a user-defined menu after any of the actions listed above, you must re-execute the instructions that created the menu.

The calculator has two program instructions for clearing the function-key definitions.

### Why Clear the Function Keys?

There are several reasons to clear the function keys in a program.

- ▶ You may want to define a function key for a limited time. By clearing the function key, you erase the function description from the display and cancel the function definition.
- ▶ If you use more than one menu in the program, you may need to clear some portion of the menu.

### Example

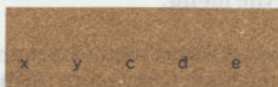
For example, suppose you want to display the following menus at different points in a program. The first menu has five options.



The second menu has two options.



When defining the second menu, you cannot redefine just the **F1** and **F2** keys. You must also clear **F3**, **F4**, and **F5**. Otherwise, the second menu would display:



You can list the addresses and names of labels used in program memory by using the **LIST** (LST) function.

## How to Clear Function Keys

The calculator has instructions to clear function-key definitions either individually or all at once.

- In the previous example, it is more efficient to clear all the function keys before you redefine **[F1]** and **[F2]**.
- In a case where two menus are almost identical, except that the second menu has one or two fewer selections, it is more efficient to clear the function keys individually.

Use one of the following procedures within a program when you want to clear the function-key definitions.

- To clear all five definitions at once, use

**[2nd] [DFN] [CLEAR]**

The mnemonic for **[2nd] [DFN] [CLEAR]** is DFN CLR.

- To clear the definition of a specific function key, use

**[2nd] [DFN] Fx [CLEAR]**

where *Fx* represents the function key (**[F1]**–**[F5]**) that you want to clear.

The mnemonic for **[2nd] [DFN] Fx [CLEAR]** is DFN Fx CLR.

## Listing Program Labels

---

You can list the addresses and names of labels used in program memory by using the **LIST** <LBL> function.

---

**Listing Program Labels** To list the labels in the program currently in program memory:

1. Press **LIST** <LBL>.

The display shows:



START LISTING AT  
1st PC

<1st> Begins searching for labels at address 0000.

<PC> Begins searching for labels at the address specified by the current setting of the program counter.

2. Select the point at which you want to begin the listing. Unless you pause or stop the listing, the calculator lists through the last label in program memory.

The listing format shows the step address of the label followed by the label name. If you have a printer connected, the listing is also printed.

**Controlling the Speed of the Listing**

If you do not have a printer connected, the calculator pauses for one second before displaying the next label. You can use the **→** key to pause the listing indefinitely or to advance to the next label, as described on page 1-14 of this guide.

**Stopping the Listing**

To stop a listing before it has finished, hold down the **BREAK** or **HALT** key until **LIST:** reappears in the display.

The **ASM** (assemble) function can increase the speed of programs that perform frequent transfers by label. Assembled programs execute faster because the calculator does not have to search for a label before transferring control.

## Assembling a Program

The **[2nd] [ASM]** (assemble) key sequence, executed as a keyboard command or program instruction, converts all program label references to the step addresses of those labels.

- ▶ GTL instructions are converted to GTO instructions.
- ▶ SBL instructions are converted to SBR instructions.
- ▶ DFN instructions are converted to DFA (define absolute) instructions. (You cannot enter the DFA instruction from the keyboard.)

### Before assembly

### After assembly

```
0000 LBL XX CLR 20
0006 STO 020
0009 LBL YY INC 020 39
0017 IF< 020 GTL ZZ CLR
0024 STO IND 020 GTL YY
0031 LBL ZZ CLR STO 020
0038 DFN F1:ENT@XX HLT
```

```
0000 LBL XX CLR 20
0006 STO 020
0009 LBL YY INC 020 39
0017 IF< 020 GTO 0034 CLR
0024 STO IND 020 GTO 0012
0031 LBL ZZ CLR STO 020
0038 DFA F1:ENT@0003 HLT
```

To assemble a program currently stored in program memory, press **[2nd] [ASM]**.

## Disassembling a Program

Disassembling a program restores all references to labels in the program. You can then modify the program without the editing difficulties associated with using numeric transfer addresses.

To disassemble the program currently stored in program memory, press **[INV] [2nd] [ASM]**.

Use this section as a source of reference information on changing the order of program execution within a program. For information on transferring execution to a program stored in the calculator's file space or in a Constant Memory™ cartridge, refer to the "File Operations" chapter of this guide.

**Label** **[2nd] [LBL] aa**—Labels a program segment. Labels are used in a program to mark locations for transfer functions. **[2nd] [LBL]** is ignored as a keyboard command.

**List Labels** **[LIST] <LBL>**—Lists labels used in program memory. The listing format shows the program address followed by the label name. As a keyboard command, **<LBL>** lets you start the label search for the list at the beginning of program memory or at the current location of the program counter. As a program instruction, **<LBL>** always starts the label search for the list at program address 0000.

The instruction mnemonic for **[LIST] <LBL>** is LL.

**Go To Label** **[2nd] [GTL] aa**—As a program instruction, **[2nd] [GTL]** transfers execution to the instruction following label *aa* in the current program or file. As a keyboard command, **[2nd] [GTL]** sets the program counter to the instruction following label *aa* in program memory, but does not start program execution.

**Go To** **[INV] [2nd] [GTL] nnnn**—As a program instruction, **[INV] [2nd] [GTL]** transfers execution to program step *nnnn* in the current program or file. As a keyboard command, **[INV] [2nd] [GTL]** sets the program counter to step *nnnn* in program memory, but does not start program execution.

The instruction mnemonic for **[INV] [2nd] [GTL]** is GTO.

**Setting the Program Counter** You can use **[2nd] [GTL]** and **[INV] [2nd] [GTL]** to set the program counter to a desired location before you enter the learn mode. For example, if you want to edit step 0025 of program memory, first press **[INV] [2nd] [GTL] 0025** to set the program counter to step 0025. Then press **[LEARN] <PC>** to enter the learn mode.

**The Subroutine Return Stack** Subroutine return addresses are stored in a system memory area called the subroutine return stack. Each time an SBL or SBR instruction is executed, a return address is added to the stack. Each time a return instruction is executed, the return address that was stored last is removed from the stack.

The calculator can store up to eight return addresses. If a program exceeds this limit, the error message **SBR STACK FULL** is displayed and the program stops. Any error that stops program execution clears the subroutine return stack.

**Subroutine Label** **[2nd] [SBL] aa**—As a program instruction, **[2nd] [SBL]** stores the address of the next instruction as a return address and then transfers execution to the instruction following label *aa* in the current program or file. As a keyboard command, **[2nd] [SBL]** starts execution at the labeled segment in program memory, but does not store a return address. Using **[2nd] [SBL]** as a keyboard command clears the subroutine return stack.

**Subroutine** **[INV] [2nd] [SBL] nnnn**—As a program instruction, **[INV] [2nd] [SBL]** stores the address of the next instruction as a return address and then transfers execution to the program segment beginning at address *nnnn* in the current program or file. As a keyboard command, **[INV] [2nd] [SBL]** transfers execution to address *nnnn* in program memory, but does not save a return address. Using **[INV] [2nd] [SBL]** as a keyboard command clears the subroutine return stack.

The instruction mnemonic for **[INV] [2nd] [SBL]** is SBR.

(continued)

**Return** **[2nd] [RTN]**—Transfers program execution to the return address stored most recently. If no return addresses are stored in the subroutine return stack, a return instruction stops program execution just as a halt instruction does.

As explained in the “File Operations” chapter of this guide, you can execute an entire program as a subroutine after the program is saved as a file. If you intend to execute a program as a subroutine, use a RTN instruction at the end of the program instead of a HLT instruction.

**Define Function Key** **[2nd] [DFN] Fx:aaa@aa**—Defines function key *Fx*, where *Fx* represents one of the five function keys, *aaa* represents the three-character message to be displayed above the function key, and *aa* represents the label to which execution transfers when you press *Fx*. The calculator supplies the “:” and “@” characters as the instruction is entered into program memory.

The function keys are activated after program execution is stopped and remain in effect as long as the definitions are visible in the display. If subsequent system-menu operations replace the definitions, you can restore the most recently defined function keys by pressing **[OLD]**.

**[2nd] [DFN]** is ignored as a keyboard command.

**Clear All Function Keys** **[2nd] [DFN] [CLEAR]**—Clears the definitions of all the function keys and erases the function labels from the display. After function definitions have been cleared by this sequence, they cannot be recalled by **[OLD]**.

**[2nd] [DFN] [CLEAR]** is ignored as a keyboard command.

The instruction mnemonic for **[2nd] [DFN] [CLEAR]** is DFN CLR.

---

### Clear a Function Key

**[2nd] [DFN] Fx [CLEAR]**—Clears the definition of function key *Fx* and erases the function label from the display. Once a function definition has been cleared by this sequence, it cannot be recalled by **[OLD]**. **[2nd] [DFN] Fx [CLEAR]** is ignored as a keyboard command.

The instruction mnemonic for **[2nd] [DFN] Fx [CLEAR]** is DFN Fx CLR.

### Old Menu

**[OLD]**—Restores the most recently defined user menu and user alpha message. **[OLD]** does not restore user-defined menus that have been cleared by DFN CLR or DFN Fx CLR.

**[OLD]** does not restore system menus.

### Assemble Program

- ☐ **[2nd] [ASM]**—Changes all label addressing used by the program in program memory into direct numeric addressing. Assembling converts GTL instructions to GTO, SBL instructions to SBR, and DFN instructions to DFA. You cannot enter the DFA (define function absolute) instruction from the keyboard.

### Disassemble Program

**[INV] [2nd] [ASM]**—Disassembles a previously assembled program. This restores label addresses and converts GTO instructions to GTL, SBR instructions to SBL, and DFA instructions to DFN. Instructions that were stored originally with numeric addressing are not converted to label addressing.



# Chapter 5: Using Tests in a Program

---

This chapter shows you how to use tests in a program. By using a test instruction, you can design a program that “decides” for itself whether to execute an instruction.

---

## Table of Contents

Introduction .....	5-2
Using Comparison Tests .....	5-4
Using DSZ Tests .....	5-7
Using the YES/NO Test .....	5-11
Using Flags .....	5-13
Ways to Use Tests with Transfer Instructions .....	5-16
Using a Test to Exit a Loop .....	5-22
Reference Section .....	5-25

A test enables a program to skip the execution of an instruction, depending on some condition in the program. For example, you can design a program to execute a particular instruction only if the result of a calculation is greater than 100.

---

### Types of Tests

You can use four types of tests in your programs. These tests are accessed through the **TESTS** and **FLAGS** keys.

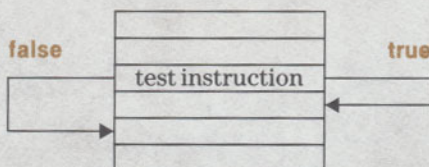
- ▶ Comparison Test—Compares the value in the numeric display register with the value in a data register.
- ▶ DSZ Test—Decrements (or increments) the value in a data register and then tests if the value is zero.
- ▶ Flag Test—Tests the status of a flag to determine if it is set or reset.
- ▶ YES/NO Test—Lets you control the execution of a program by responding to a yes/no question.

### How a Test Affects Program Execution

When using a test, you specify a condition to test. The program tests that condition and determines if the result of the test is true or false.

- ▶ If the result is true, program execution continues in normal sequential order.
- ▶ If the result is false, the program skips the first instruction that follows the test.

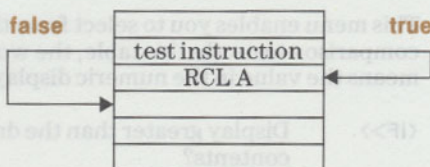
This rule is summarized as “Do if true, skip if false.”



Because a test determines whether the next program instruction is executed, you can perform tests only in a program—not from the keyboard.

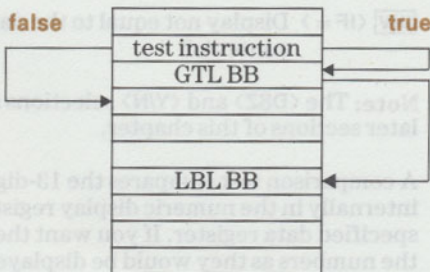
## Using a Test

For example, you may want to use a test to determine whether to execute a memory operation such as RCL A. This case is illustrated below. If the result of the test is true, the RCL A instruction is executed. If the result is false, the RCL A instruction is skipped.



Although a test skips only one instruction, you can make a program skip or execute entire segments by using a transfer instruction following the test.

For example, you may want a program to transfer to a segment labeled BB if a certain condition is true. This case is illustrated below. If the result of the test is true, the GTL BB instruction is executed. If the result is false, the GTL BB instruction is skipped.



If the instruction following a test is an inverse function such as INV SIN, only INV is skipped.

## Using Comparison Tests

---

You can use one of six comparison tests to compare the value in the numeric display register with the contents of a data register. (The value in the numeric display register is the 13-digit number used internally, not necessarily the rounded value shown in the display.)

---

### Available Comparisons

When you press **TESTS**, the following menu is displayed.



TEST FUNCTIONS  
IF> IF< IF= DSZ Y/N

This menu enables you to select from the following comparison tests. (In this table, the word “display” means the value in the numeric display register.)

- |                    |  |
|--------------------|--|
| <IF>               | Display greater than the data register contents?             |
| <b>INV</b> <IF>    | Display less than or equal to the data register contents?    |
| <IF<               | Display less than the data register contents?                |
| <b>INV</b> <IF<    | Display greater than or equal to the data register contents? |
| <IF = >            | Display equal to the data register contents?                 |
| <b>INV</b> <IF = > | Display not equal to the data register contents?             |

**Note:** The <DSZ> and <Y/N> selections are described in later sections of this chapter.

A comparison test compares the 13-digit numbers stored internally in the numeric display register and the specified data register. If you want the test to compare the numbers as they would be displayed, use <RND> to round them to the current display format.

## Performing a Comparison Test

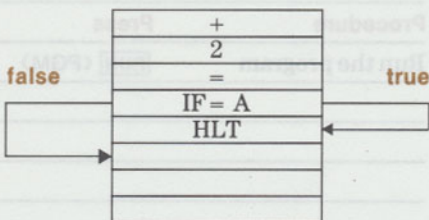
To include a comparison test in a program, use one of the following key sequences:

- ▶ **TESTS** <IF> nnn or X
- ▶ **TESTS** **INV** <IF> nnn or X
- ▶ **TESTS** <IF< nnn or X
- ▶ **TESTS** **INV** <IF< nnn or X
- ▶ **TESTS** <IF = > nnn or X
- ▶ **TESTS** **INV** <IF = > nnn or X

For example, suppose that you want to add 2 to the contents of the numeric display register and then determine if the result is equal to the contents of data register A. If the result is equal to the value in register A, you want the program to stop.

The key sequence to enter this test in a program is:

**+ 2 = TESTS <IF = > A HLT**



- ▶ If the result of the addition is equal to the value in register A, the test result is true and the HLT instruction is executed.
- ▶ If the result of the addition is not equal to the value in register A, the test result is false and the HLT instruction is skipped.

(continued)

## Using Comparison Tests (Continued)

**Example** Program loops provide a good example of the advantages of using tests in a program. Write a program that counts by twos, like the example shown on page 4–9, but uses a test to stop the loop when the count reaches 20.

PC =	Program Mnemonics	Comments
0000	20 STO A	Stores comparison value
0004	CLR	Clears calculator
0005	LBL LL	Begins loop
0008	+ 2 =	Adds 2
0011	PAU	Pauses to show result
0012	IF = A	Does count = 20?
0014	HLT	Yes—stop the program
0015	GTL LL	No—repeat loop

### Running the Example

Run the program and observe the result.

Procedure	Press	Display
Run the program	<b>RUN</b> <PGM>	2.
		4.
		:
		20.

(continued)

A DSZ test combines two operations into a single instruction. First, it decrements the value in a specified data register. Then, it tests whether the value has reached zero. A DSZ test is particularly useful to limit the number of times a loop repeats.

### Available DSZ Tests

When you press **TESTS**, you can select one of two DSZ tests.

TEST FUNCTIONS  
IF> IF< IF= DSZ Y/N

<DSZ> Decrement and skip if zero.

**INV** <DSZ> Decrement and execute if zero.

### Rules Used in Decrementing

In a DSZ test, “decrement” means that the value in the data register approaches zero, based on the following rules.

- ▶ If the stored value is greater than 1, the DSZ test subtracts 1 from the value.
- ▶ If the stored value is less than -1, the DSZ test adds 1 to the value.
- ▶ If the value is between -1 and 1, the DSZ test stores a zero in the register.

### Performing a DSZ Test

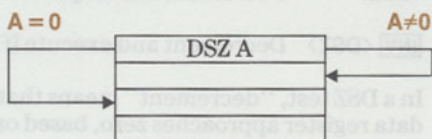
To include a DSZ test in a program, use one of the following key sequences:

- ▶ **TESTS** <DSZ> *nnn* or *X*
- ▶ **TESTS** **INV** <DSZ> *nnn* or *X*

(continued)

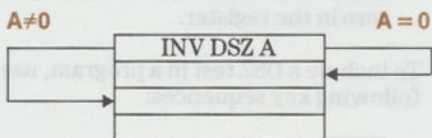
### Decrement and Skip if Zero

The “decrement and skip if zero” test decrements the value in a specified data register and then tests if the register value equals zero. If the register value does equal zero, the program skips the instruction that immediately follows the DSZ test. If the register value does not equal zero, execution continues in normal sequential order.



### Decrement and Execute if Zero

The “decrement and execute if zero” test decrements the value in a specified data register and then tests if the register value equals zero. If the register value does not equal zero, the program skips the instruction that immediately follows the INV DSZ test. If the register value does equal zero, execution continues in normal sequential order.



**Example**

The DSZ tests offer you a convenient and accurate method of executing a program sequence a predetermined number of times. By using either DSZ or INV DSZ, you can control whether the instruction following the test is executed a specific number of times or skipped a specific number of times.

The following program illustrates the operation of a DSZ test in a loop. The program allows you to enter the initial register value for the DSZ test. It then displays the contents of that data register each time the loop executes.

PC =	Program Mnemonics	Comments
0000	STO C	Stores initial DSZ value
0002	LBL LL	Begins loop
0005	RCL C	Recalls DSZ value
0007	PAU	Displays value
0008	DSZ C	Decrements and tests C
0010	GTL LL	C≠0—repeat loop
0013	HLT	C=0—stop program

(continued)

## Running the Example

Enter 4 into the display and run the program.

Procedure	Press	Display
Display RUN menu	<b>RUN</b>	SELECT:
Enter count value	4	4
Run the program	<PGM>	4.
		3.
		2.
		1.

Notice that 0 is not displayed. When the data register is decremented to zero, the program exits the loop and stops the program.

The following table shows what appears in the display if you run the program with a negative number and with a number that contains a fractional part.

Starting with - 4	Starting with 4.3
- 4.	4.3
- 3.	3.3
- 2.	2.3
- 1.	1.3
	0.3

The Y/N test gives you an easy way to let the user control the program by answering yes or no to a question. Unlike the tests discussed previously in this chapter, the Y/N test does not compare any numeric values in the program.

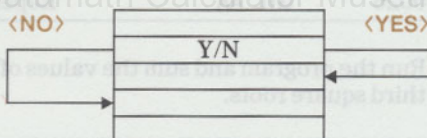
### The YES/NO Test

When you press **TESTS**, you can enter the Y/N test into your program.

TEST FUNCTIONS  
IF> IF< IF= DSZ Y/N

<Y/N> Defines the **F1** key as YES and the **F2** key as NO and halts the program. This lets the user of your program respond to an alpha message you have placed in the display.

If you press <NO> in response to the Y/N test, the first instruction following the test is skipped. If you press <YES> in response to the Y/N test, execution continues in normal sequential order.



Although you could create a yes/no menu using the DFN instruction, the Y/N test has the following advantages whenever you want a program to pose a yes/no question.

- ▶ You do not have to include <YES> and <NO> function key definitions.
- ▶ You do not have to include a HLT instruction in the program.
- ▶ You do not have to clear any function key definitions. (The calculator clears the definitions automatically.)

(continued)

## Example

The following program creates a loop that calculates the square roots of a sequence of numbers beginning with 1. The program uses a Y/N test to ask if you want to sum each result to data register B.

PC =	Program Mnemonics	Comments
0000	0 STO A STO B	Initializes A and B
0005	LBL AA	Begins loop
0008	INC A	Increments count
0010	RCL A	Gets current count
0012	SQR	Calculates square root
0013	PAU	Displays square root
0014	'SUM TO B?'	Creates message
0023	Y/N	YES/NO test
0024	ST+ B	<YES>—sum to B
0026	GTL AA	Repeats loop

## Running the Example

Run the program and sum the values of the first and third square roots.

Procedure	Press	Display
Start the program	<b>[RUN]</b> <PGM>	1.
		SUM TO B?
Sum 1st number	<YES>	1.414213562
		SUM TO B?
Don't sum 2nd number	<NO>	1.732050808
		SUM TO B?
Sum 3rd number	<YES>	2.
		SUM TO B?
Check the sum	<b>[RCL]</b> B	2.732050808
Clear function keys	<b>[2nd]</b> <b>[F:CLR]</b>	2.732050808

Flags give you a way to retain information about the state of the calculator or the status of a variable for later use. For example, you might set a flag if the value of an intermediate result is negative. Later, you can test the flag to see if the number was negative, even though the number itself is no longer available.

---

### What Is a Flag?

You can think of a flag as a two-position switch that is either set or reset. You can:

- ▶ Set a flag.
- ▶ Reset a flag.
- ▶ Test whether a flag is set.
- ▶ Test whether a flag is reset.

After you set or reset a user flag, that setting is retained until you change it. Although you can set or reset flags from the keyboard or in a program, you can test a flag only in a program.

### User Flags

The TI-95 has 16 user flags, numbered 00 through 15, that you can use. Additional flags used by the calculator and by library cartridges are discussed in Appendix C.

Flags 00–14 can be used as needed in a program.

Flag 15 is the halt-on-error flag. When this flag is set, any error condition that occurs while a program is running causes the program to halt.

When flag 15 is reset (the default condition), the following errors do not stop the program.

- ▶ OVERFLOW
- ▶ INVALID ARGUMENT
- ▶ AOS STACK FULL
- ▶ I/O ERROR *nnn*

By executing the LS (list status) instruction and then examining the numeric display register, the program can test whether any of these errors has occurred and can handle the error as you prefer.

(continued)

## The Flag Functions Menu

When you press **FLAGS**, the following menu is displayed.

FLAG FUNCTIONS  
CLR SF RF TF

This menu enables you to select the following flag instructions.

<CLR> Clears (resets) user flags 00–14. The instruction mnemonic for this is CFG.

<SF> Sets a specified flag.

<RF> Resets a specified flag.

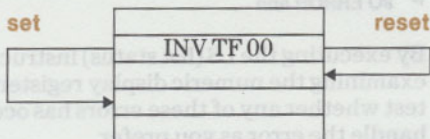
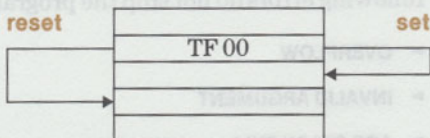
<TF> Tests whether a specified flag is set.

**INV** <TF> Tests whether a specified flag is reset

The <SF>, <RF>, <TF>, and **INV** <TF> instructions are followed by a two-digit numeric field that specifies the flag. For example, <SF> 01 sets flag 01.

## Testing a Flag

A test flag instruction affects program control in the same manner as a comparison or DSZ test.



### Example

The following program uses flag 00 to record your response to a Y/N test. The program then executes a loop that sums entered values to register A. On each pass through the loop, the program tests the flag to determine whether it should display the subtotal. Notice that the example uses CFG at the beginning of the program to make sure all flags are in a known state.

PC =	Program Mnemonics	Comments
0000	0 STO A	Initializes register A
0003	CFG	Clears flags
0004	'DISPLAY SUM?'	Creates message
0016	Y/N	YES/NO test
0017	SF 00	<YES>—set flag 00
0019	CLR	Clears message
0020	LBL BB	Begins loop
0023	BRK	Waits for entry
0024	ST + A	Sums entry to A
0026	TF 00	Is flag 00 set?
0028	RCL A	Yes—display subtotal
0030	GTL BB	Repeats the loop

### Running the Example

Run the program and display the intermediate results.

Procedure	Press	Display
Start the program	<b>RUN</b> <PGM>	DISPLAY SUM?
Specify YES	<YES>	0.
Enter values	25 <GO>	25.
	50 <GO>	75.
	6 <GO>	81.

## Ways to Use Tests with Transfer Instructions

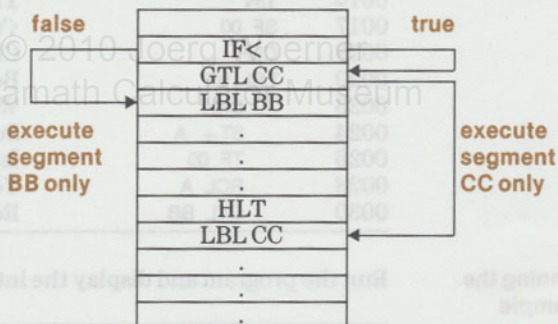
By following a test with a transfer instruction such as GTL or SBL, you can use the test to conditionally execute entire program segments. You can use such a combination, for example, to conditionally execute one of two segments or to conditionally call a subroutine.

### Executing One of Two Segments

You may want to execute only one of two program segments according to the result of a test.

In the following illustration, an IF< test is used with a GTL transfer instruction to determine which of the two segments BB or CC will be executed.

- If the test is true, the program skips the segment labeled BB and transfers control to label CC.
- If the test is false, the program executes the segment labeled BB. Because the BB segment ends with a HLT instruction, the program does not execute label CC.



### Example

Write a program that uses two labeled program segments: one to sum a negative number into data register B, and a second to sum a positive number into register C. A test of the number determines which segment is executed.

PC =	Program Mnemonics	Comments
0000	STO A	Stores entered number
0002	0 IF< A	Is number positive?
0005	GTL CC	Yes—go to label CC
0008	LBL BB	No—execute label BB
0011	'NEGATIVE' PAU	Creates message
0020	RCL A	Recalls number
0022	ST+ B RCL B	Sums to B and recalls sum
0026	HLT	Stops program
0027	LBL CC	Labels segment
0030	'POSITIVE' PAU	Creates message
0039	RCL A	Recalls number
0041	ST+ C RCL C	Sums to C and recalls sum
0045	HLT	Stops program

### Running the Example

Test the program by entering a negative number and a positive number.

Procedure	Press	Display
Clear display and data registers	<b>CLEAR</b> <b>2nd</b> [CMS]	0.
Display RUN menu	<b>RUN</b>	SELECT:
Enter a negative number	6.4 <b>+/-</b> <PGM>	NEGATIVE -6.4
Enter a positive number	10 <PGM>	POSITIVE 10.

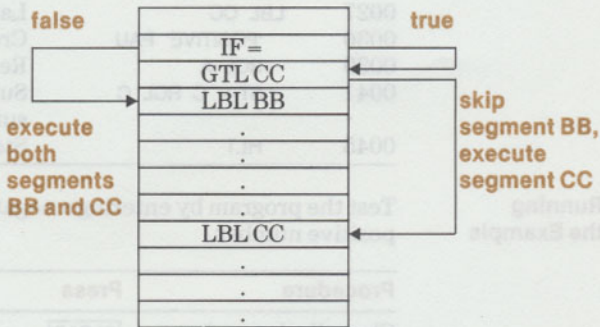
(continued)

### Skipping a Segment of a Program

You may want to skip a segment of a program depending on the result of a test. For example, if the result of a calculation is negative, you may want the program to skip the next five instructions.

In the following illustration, an IF = comparison test is used with a GTL instruction to determine whether the segment labeled BB will be skipped. (The main difference between this and the previous example is the removal of the HLT instruction that preceded label CC.)

- ▶ If the test is true, the program skips the segment labeled BB and transfers control to label CC.
- ▶ If the test is false, the program executes the segment labeled BB and continues executing through the segment labeled CC.



### Example

This program sums each number you enter to data register A. The segment labeled BB sums a number to data register B, but is executed only for numbers that are evenly divisible by 6.

PC =	Program Mnemonics	Comments
0000	CMS	Clears data registers
0001	DFN F1:ENT@EN	Defines F1 for entry
0008	LBL AA	Labels segment
0011	CLR	Clears calculator
0012	'ENTER NUMBER'	Creates message
0024	HLT	Waits for entry
0025	LBL EN	Labels segment
0028	ST+ A STO C	Sums to A, stores in C
0032	/ 6 =	Divides number by 6
0035	FRC STO D	Stores fractional part
0038	0 INV IF= D	Is number non-zero?
0042	GTL CC	Yes—go to to CC
0045	LBL BB	Labels segment
0048	'DIVISIBLE BY 6'	Creates message
0062	PAU	Pauses
0063	RCL C ST+ B	Sums original number to B
0067	LBL CC	Labels segment
0070	'SUM = '	Creates message
0074	COL 16 MRG A PAU	Shows the sum
0079	GTL AA	Repeats loop

### Running the Example

Run the program and enter a number that is not divisible by 6. The program sums the number to register A, but skips label BB and does not sum the number to register B.

Now enter a number that is divisible by 6. The program sums the number to register A, displays an additional message, and sums the number to register B.

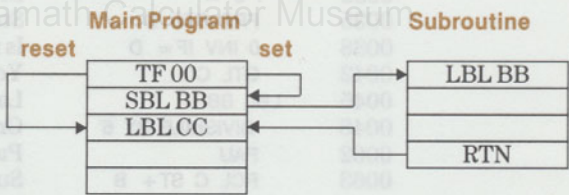
(continued)

Conditional Execution of a Subroutine

Your program may have a subroutine that you want to call only under specific conditions. By preceding an SBL or SBR instruction with a test instruction, you can use the test to control whether the subroutine will be called.

In the following illustration, a TF test is used with SBL to determine whether the program will execute the subroutine labeled BB.

- ▶ If flag 00 is set, the program executes subroutine BB before returning control to the segment labeled CC.
- ▶ If flag 00 is reset, the program proceeds to the segment labeled CC without executing the subroutine.



### Example

This program contains a subroutine labeled BB that displays the subtotal of numbers as you sum them to data register A. The test instruction at program step 0048 ensures that the subroutine is called only if you select <YES> at the start of the program.

PC =	Program Mnemonics	Comments
0000	CFG	Clears flags
0001	0 STO A STO C	Clears registers A and C
0006	'SHOW SUBTOTAL?'	Creates message
0020	Y/N	Want subtotal?
0021	SF 00	<YES>—set flag 00
0023	DFN CLR	Clears Y/N definitions
0025	LBL AA	Labels segment
0028	INC C	Increments item count
0030	CE 'ENTER ITEM'	Creates message
0041	COL 14 MRG C	Merges item count
0045	BRK	Waits for entry
0046	ST+ A	Sums entry to A
0048	TF 00	Subtotal requested?
0050	SBL BB	Yes—show subtotal
0053	GTL AA	Repeats loop
0056	LBL BB	Labels subroutine BB
0059	'TOT = '	Creates message
0063	COL 16 MRG A	Merges subtotal
0067	PAU	Pauses
0068	RTN	Returns

### Running the Example

Run the program and select <YES> when it asks if you want to show the subtotal. After the program prompts you for each item to be summed, subroutine BB shows the current subtotal.

Run the program a second time and select <NO>. The program does not call subroutine BB to display the current subtotal as you enter numbers.

## Using a Test to Exit a Loop

You can use a test instruction within a loop as a means of terminating the loop. You can design a loop so that it terminates after a specified number of repetitions, when a specified condition exists, or when either of those two events occurs.

**Exiting on a Specified Count** A DSZ or INV DSZ test is commonly used to repeat a loop a specified number of times.

**Example** Write a program that calculates the average of numbers that you enter. Design the program so that you can specify how many numbers you want to average.

PC =	Program Mnemonics	Comments
0000	0 STO A	Clears sum register
0003	'# OF ENTRIES?'	Creates message
0016	BRK	Stops for input
0017	STO B	Stores # of entries
0019	STO C	Stores loop count
0021	LBL AA	Labels segment
0024	'ENTER NUMBER'	Creates message
0036	BRK	Stops for input
0037	ST+ A	Sums entry
0039	DSZ C	Does count = 0?
0041	GTL AA	No—repeat loop
0044	RCL B ST/ A	Yes—calculate average
0048	RCL A	Recalls average
0050	HLT	Stops program

**Running the Example** Run the program and calculate the average of 121 and 9.

Procedure	Press	Display
Run the program	<b>RUN</b> <PGM>	# OF ENTRIES?
Enter the count	2 <GO>	ENTER NUMBER
Enter first number	121 <GO>	ENTER NUMBER
Enter second number	9 <GO>	65.

### Exiting on a Specified Condition

In many instances, you want to exit a loop only if a specified condition exists. For example, you may want to repeat a loop until a variable in the program is equal to zero.

### Example

The following program raises values you enter to the fourth power. The program repeats the loop indefinitely until you enter a value of 0.

PC =	Program Mnemonics	Comments
0000	LBL AA	Begins loop
0003	'ENTER VALUE'	Creates message
0014	BRK	Waits for entry
0015	STO A	Stores for comparison
0017	0 IF = A	Does entry = 0?
0020	HLT	Yes—terminate loop
0021	RCL A $y^x$ 4 =	No—calculate power
0026	PAU PAU	Pauses two seconds
0028	GTL AA	Repeats loop

### Running the Example

Enter the program and run it.

Procedure	Press	Display
Start the program	<b>RUN</b> <PGM>	ENTER VALUE
Enter a value	2.3 <GO>	27.9841
Enter a value	1.89 <GO>	12.75989841
Enter terminating value	0 <GO>	0.

(continued)

### Exiting on Count or Condition

You may want to design a loop that terminates when either a specified count is reached or some other condition occurs. Such a loop must contain more than one test.

The following program calculates the fourth power of numbers you enter. The program combines the methods used in the two previous examples for exiting from a loop.

- ▶ It uses a loop counter to limit the number of entries to a maximum of five.
- ▶ It tests for an entered value of 0. This lets you terminate the loop before you have entered all five values.

PC =	Program Mnemonics	Comments
0000	5 STO A	Stores the loop counter
0003	LBL AA	Begins loop
0006	'ENTER VALUE'	Creates message
0017	BRK	Waits for entry
0018	STO B	Stores for comparison
0020	0 IF = B	Does entry = 0?
0023	HLT	Yes—terminate loop
0024	RCL B $y^x$ 4 =	No—calculate power
0029	PAU PAU	Pauses two seconds
0031	DSZ A	Does count = 0?
0033	GTL AA	No—repeat loop
0036	HLT	Yes—terminate loop

### Running the Example

Run the program and enter five numbers. The program terminates after it calculates the fourth power of the fifth number.

Run the program a second time and enter 0 as one of the numbers. The program terminates when you enter the 0.

Use this section as a source of reference information on the test and flag instructions.

---

### Test Functions

**TESTS**—Displays the tests menu. Test functions can be executed only as program instructions.

**TESTS** <IF>> *nnn* or *X*—Tests if the value in the numeric display register is greater than the value in data register *nnn* or *X*. If the test result is true, program execution proceeds normally. If the test result is false, the instruction following the test is skipped.

**TESTS** **INV** <IF>> *nnn* or *X*—Tests if the value in the numeric display register is less than or equal to the value in data register *nnn* or *X*. If the test result is true, program execution proceeds normally. If the test result is false, the instruction following the test is skipped.

**TESTS** <IF<> *nnn* or *X*—Tests if the value in the numeric display register is less than the value in data register *nnn* or *X*. If the test result is true, program execution proceeds normally. If the test result is false, the instruction following the test is skipped.

**TESTS** **INV** <IF<> *nnn* or *X*—Tests if the value in the numeric display register is greater than or equal to the value in data register *nnn* or *X*. If the test result is true, program execution proceeds normally. If the test result is false, the instruction following the test is skipped.

**TESTS** <IF = > *nnn* or *X*—Tests if the value in the numeric display register is equal to the value in data register *nnn* or *X*. If the test result is true, program execution proceeds normally. If the test result is false, the instruction following the test is skipped.

**TESTS** **INV** <IF = > *nnn* or *X*—Tests if the value in the numeric display register is not equal to the value in data register *nnn* or *X*. If the test result is true, program execution proceeds normally. If the test result is false, the instruction following the test is skipped.

(continued)

## Test Functions (Continued)

**TESTS** <DSZ> *nnn* or *X*—Decrements the value in data register *nnn* or *X* (or increments the value if it is negative). After decrementing the register value, the calculator tests if the value equals zero. If the value is equal to zero, the instruction following the test is skipped. If the value is not equal to zero, execution proceeds normally.

The rules for decrementing the register value are: values greater than 1 are decreased by 1, values less than -1 are increased by 1, and values between -1 and 1 are set to zero.

**TESTS** **INV** <DSZ> *nnn* or *X*—Decrements the value in data register *nnn* or *X* (or increments the value if it is negative). After decrementing the register value, the calculator tests if the value equals zero. If the value is equal to zero, execution proceeds normally. If the value is not equal to zero, the instruction following the test is skipped.

The rules for decrementing the register value are: values greater than 1 are decreased by 1, values less than -1 are increased by 1, and values between -1 and 1 are set to zero.

**TESTS** <Y/N>—Defines **F1** as <YES> and **F2** as <NO> and stops program execution. If <YES> is pressed, execution proceeds normally. If <NO> is pressed, the instruction following the <Y/N> instruction is skipped.

**Note:** The <Y/N> function definitions are treated like a user-defined menu by the calculator. Therefore, pressing **OLD** will restore the <Y/N> menu after it has been cleared by a system menu. To clear the <Y/N> definitions in a program, use the **2nd** **[DFN]** **CLEAR** or **2nd** **[DFN]** **Fx** **CLEAR** key sequence.

(continued)

---

## Flag Functions

**FLAGS**—Displays the flags menu. The calculator has 100 flags. Flags 00–14 are general-purpose flags that can be used as desired in a program. Flag 15 is the halt-on-error flag. Flags 16–23 are reserved for use by programs in the software library cartridges. Flags 24–99 are system flags. Appendix C lists the system flags used by the calculator.

**FLAGS** <CLR>—Clears (resets) user flags 00–14. The instruction mnemonic for **FLAGS** <CLR> is CFG.

**FLAGS** <SF> *nn*—Sets flag *nn*. Flags 00–15 can be set from the keyboard or within a program. Unless the calculator is in the unprotected mode, flags 16–99 can be set only from within a program. (Unprotected mode is described in Appendix A.)

**FLAGS** <RF> *nn*—Resets flag *nn*. Flags 00–15 can be reset from the keyboard or within a program. Unless the calculator is in the unprotected mode, flags 16–99 can be reset only from within a program.

**FLAGS** <TF> *nn*—Tests if flag *nn* is set. If flag *nn* is set, execution proceeds normally. If flag *nn* is reset, the instruction following the test is skipped. This function can be executed only as a program instruction.

**FLAGS** <INV> <TF> *nn*—Tests if flag *nn* is reset. If flag *nn* is reset, execution proceeds normally. If flag *nn* is set, the instruction following the test is skipped. This function can be executed only as a program instruction.



## Chapter 6: Using Indirect Addressing

---

This chapter introduces the concept of indirect addressing and discusses ways you can use this feature in your programs.

---

<b>Table of Contents</b>	Introduction .....	6-2
	Indirect Data Register Operations .....	6-4
	Indirect Transfer of Program Control .....	6-8
	Other Indirect Operations .....	6-13
	Reference Section .....	6-14

© 2010 Joerg Arnecke  
Gammath-Calculator-Museum

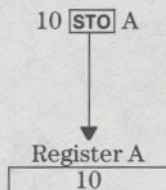
Indirect addressing gives you an alternate way to specify a field.

### What Is Indirect Addressing?

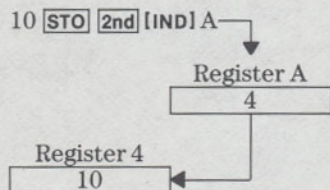
Up to now, you have supplied a specific field whenever you used a calculator function that required a field. For example, to store 10 in data register A, you used the key sequence 10 **[STO]** A. This form of specifying a field is called **direct addressing**, because you directly specify the field.

**Indirect addressing** is an alternate method of specifying a field. In indirect addressing, you do not specify the actual field, but a data register where the field is stored. For example, the key sequence 10 **[STO]** **[2nd]** **[Ind]** A instructs the calculator to store 10 in the data register specified by the contents of register A. The difference between direct and indirect addressing is illustrated below.

#### Direct Addressing



#### Indirect Addressing



In the indirect example, the contents of register A determine where the value 10 is stored. In this case, register A contains 4, so 10 is stored in register 004.

---

## Using Indirect Addressing

To specify indirect addressing, substitute the key sequence

**[2nd] [IND] nnn** or **X**

for the field of the instruction you want to address indirectly. The data register following the **[2nd] [IND]** sequence is called the **pointer register**.

To avoid causing errors when using indirect addressing, apply the following rules.

- ▶ If the instruction you are addressing indirectly uses a numeric field, be sure that the pointer register contains a numeric value that is a valid numeric field. The integer value of the contents of the pointer register is used as the numeric field. (Negative values are interpreted as zero.)
- ▶ If the instruction you are addressing indirectly uses an alpha field, be sure that the pointer register contains alpha characters that are valid as an alpha field. The calculator uses the leftmost characters stored in the pointer register as the alpha field. Any extra characters stored in the pointer register are ignored.

## Restrictions

You can use indirect addressing with any instruction that has a field, except the three instructions listed below. A complete list of the instructions that require a field is given in Appendix C.

- ▶ LBL (Label segment)
- ▶ DFN (Define function key)
- ▶ DFA (Define function key absolute)

## Indirect Data Register Operations

You can use indirect addressing with any data register operation.

### Indirect Storing

The key sequence **[STO] [2nd] [IND] nnn** or **X** stores the value in the numeric display register in the data register specified by the contents of register *nnn* or **X**.

For example, if data register D contains 12, the key sequence 25 **[STO] [2nd] [IND]** D stores 25 in data register 012.

### Indirect Recalling

The key sequence **[RCL] [2nd] [IND] nnn** or **X** recalls the data register whose address is stored in register *nnn* or **X**.

For example, if data register D contains 12 and data register 012 contains 25, the key sequence **[RCL] [2nd] [IND]** D displays the value 25.

### Indirect Data Register Restrictions

Although you can use alpha addressing to specify the pointer register, the contents of the pointer register must be a numeric address, not an alpha address. For example, if you want to address register C indirectly, the pointer register must contain 2, not C.

### Example

Using indirect addressing, store 10 in register E. Then recall E indirectly and directly. Use register A as the pointer register.

Procedure	Press	Display
Store IND pointer	4 <b>[STO]</b> A	4.
Indirectly store 10 in register 4 (E)	10 <b>[STO] [2nd] [IND]</b> A	10.
Clear display	<b>[CLEAR]</b>	0.
Recall E indirectly	<b>[RCL] [2nd] [IND]</b> A	10.
Recall E directly	<b>[CLEAR] [RCL]</b> E	10.

### Why Use Indirect Addressing?

The power of indirect addressing may not be readily apparent when illustrated by a keyboard example, but it can be seen in programming applications. Indirect addressing enables a program to control which data register is referred to by a memory function. This is an enormous advantage when a series of similar operations must be performed on different registers.

For example, you may have a programming problem that requires the entry and storage of a series of numbers. Without indirect addressing, you would solve this problem by including a separate STO instruction for each number you want to store, as illustrated below.

STO A
BRK
STO B
BRK
STO C
BRK
STO D
BRK
.
.
.

As you can see, this method of solving the problem makes a program long and cumbersome. By using indirect addressing, you can design a loop to solve this problem, as illustrated by the example on the next page.

(continued)

**Example** Write a program that will store six data entries in registers 1 through 6. Use data register A as the pointer register.

PC =	Program Mnemonics	Comments
0000	1 STO A	Initializes IND pointer
0003	LBL ZZ	Labels segment
0006	'ENTER NUMBER'	Creates message
0018	BRK	Stops program to enter number
0019	STO IND A	Stores entry indirectly
0022	INC A	Increments IND pointer
0024	7 IF = A	Is IND pointer = to 7?
0027	HLT	Yes—stop program
0028	GTL ZZ	No—repeat the loop

**Running the Example** Use the program to store the following values: 101, 102, 103, 104, 105, 106. Check the program by listing data registers 1 through 6.

Procedure	Press	Display
Run program	<b>RUN</b>	
Enter data	< PGM>	ENTER NUMBER
	101 <GO>	ENTER NUMBER
	102 <GO>	ENTER NUMBER
	103 <GO>	ENTER NUMBER
	104 <GO>	ENTER NUMBER
	105 <GO>	ENTER NUMBER
	106 <GO>	7.
List registers	<b>LIST</b> 1	001 101.
	<REG>	002 102.
		003 103.
		004 104.
		005 105.
		006 106.
Stop listing	<b>HALT</b>	LIST:

## Indirect Memory Arithmetic

Indirect memory arithmetic lets you perform an arithmetic operation on a series of stored values or a computed register address.

The indirect memory arithmetic functions are listed below.

Key Sequence	Operation
<b>STO</b> <b>+</b> <b>2nd</b> <b>[IND]</b> <i>nnn</i> or <i>X</i>	Indirect Addition
<b>STO</b> <b>-</b> <b>2nd</b> <b>[IND]</b> <i>nnn</i> or <i>X</i>	Indirect Subtraction
<b>STO</b> <b>×</b> <b>2nd</b> <b>[IND]</b> <i>nnn</i> or <i>X</i>	Indirect Multiplication
<b>STO</b> <b>÷</b> <b>2nd</b> <b>[IND]</b> <i>nnn</i> or <i>X</i>	Indirect Division
<b>INCR</b> <b>2nd</b> <b>[IND]</b> <i>nnn</i> or <i>X</i>	Indirect Increment
<b>INV</b> <b>INCR</b> <b>2nd</b> <b>[IND]</b> <i>nnn</i> or <i>X</i>	Indirect Decrement

## Example

The following subroutine uses indirect multiplication (ST\* IND) to multiply by 3 the contents of data registers 070 through 073.

PC =	Program Mnemonics	Comments
0000	LBL XX	Label segment
0003	70 STO A	Initialize IND pointer
0007	LBL YY	Label segment
0010	3 ST* IND A	Multiply by 3 indirectly
0014	INC A	Increment IND pointer
0016	73 IF< A	Is 73 < register A?
0020	RTN	Yes—exit the subroutine
0021	GTL YY	No—repeat the loop

## Running the Example

To test the subroutine, first store known values in data registers 70 through 73. Then run the program. When the program stops, use **LIST** 70 <REG> to verify that the stored values have been multiplied by 3.

## Indirect Transfer of Program Control

---

Indirect addressing enables a transfer instruction to transfer control to a location specified by the contents of a data register.

---

### Indirect GTL

To include an indirect GTL instruction in a program, use the key sequence **[2nd] [GTL] [2nd] [IND] nnn** or **X**. Indirect GTL transfers program control to the program label stored in register **nnn** or **X**.

For example, if the leftmost two characters in data register B are AA, the instruction sequence **GTL IND B** transfers program control to label AA.

Alpha characters can be stored in a data register by using:

- ▶ The <STA> (store alpha message) function. Keep in mind that <STA> uses 10 data registers, although the indirect GTL instruction needs only the two leftmost characters of the first data register.
- ▶ The STB (store byte) function. This function is described in Appendix A.
- ▶ The STO function with the calculator in unformatted display mode. Unformatted display mode is described in Appendix A.

### Indirect GTO

To include an indirect GTO instruction in a program, use the key sequence **[INV] [2nd] [GTL] [2nd] [IND] nnn** or **X**. Indirect GTO transfers program control to the step address stored in register **nnn** or **X**.

For example, if data register D contains 444, the instruction sequence **GTO IND D** transfers program control to program address 0444.

**Indirect  
SBL**

To include an indirect SBL instruction in a program, use the key sequence **[2nd] [SBL] [2nd] [IND] nnn** or **X**. Indirect SBL calls the subroutine specified by the label stored in register *nnn* or **X**.

For example, if the leftmost two characters in data register **I** are **B1**, the instruction sequence **SBL IND I** calls the subroutine labeled **B1**.

**Indirect  
SBR**

To include an indirect SBR instruction in a program, use the key sequence **[INV] [2nd] [SBL] [2nd] [IND] nnn** or **X**. Indirect SBR calls the subroutine at the step address specified by the contents of register *nnn* or **X**.

For example, if data register **D** contains **444**, the instruction sequence **SBR IND D** calls the subroutine at program address **0444**.

**Using Indirect  
Transfers from  
the Keyboard**

Using indirect **GTL** or **GTO** as a keyboard command sets the program counter to the label or step address specified by the contents of the pointer register, but does not start execution of the program.

Using indirect **SBL** or **SBR** as a keyboard command starts program execution at the label or step address specified by the contents of the pointer register, but does not store a return address.

(continued)

**Example 1** The following program calculates the step address of one of three routines, based on a number you enter, and uses GTO IND to execute the routine.

If you use this method of indirect transfer:

- ▶ The address of the first routine must be known to the program.
- ▶ You must allocate an equal amount of memory for each of the routines to be executed. (Note the NOPs placed after the HLT instruction for routines 1 and 2.)

PC =	Program Mnemonics	Comments
0000	CLR	Clears calculator
0001	14 STO A	Stores length of routines
0005	49 STO B	Stores start of routines
0009	DFN F1:ENT@EN	Defines F1
0016	'ROUTINE (1-3)?'	Creates prompt
0030	HLT	Stops program
0031	LBL EN	Labels segment
0034	-1=	Adjusts multiplier
0037	* RCL A + RCL B =	Calculates routine address
0044	STO C	Stores routine address
0046	GTO IND C	Executes routine indirectly
0049	'ROUTINE ONE'	Routine 1 message
0060	HLT NOP NOP	End of routine
0063	'ROUTINE TWO'	Routine 2 message
0074	HLT NOP NOP	End of routine
0077	'ROUTINE THREE'	Routine 3 message
0090	HLT	End of routine

**Running  
Example 1**

When you run the program, you are prompted to identify the routine you want to execute by entering a number between 1 and 3. When you enter the number and press <ENT>, the program subtracts 1 from the number to produce a multiplier of 0, 1, or 2. The program then uses the multiplier to calculate the address of the routine and executes the routine by indirect transfer.

Procedure	Press	Display
Run the program	<b>RUN</b> <PGM>	ROUTINE (1-3)?
Specify routine 2	2 <ENT>	ROUTINE TWO
Specify routine 1	1 <ENT>	ROUTINE ONE
Specify routine 3	3 <ENT>	ROUTINE THREE
Specify nonexistent routine	4 <ENT>	INVALID ADDRESS
Clear the error	<b>CLEAR</b>	0.

**Note:** To avoid the error message or possible execution of the wrong instructions, you could use test instructions within the program to ensure the entered number is within the 1-3 limit before executing the indirect transfer.

(continued)

### Example 2

You may not want to allocate an equal amount of memory for each routine. You can overcome this limitation by creating an intermediate list of GTL instructions. The program indirectly transfers to a GTL instruction in this list, which transfers execution in turn to a final routine. The advantage of the intermediate list is that the entries in the list are all the same length. Thus, you can use the technique illustrated in Example 1 to address them indirectly. The final routines can be different lengths, as illustrated by this example.

PC =	Program Mnemonics	Comments
0000	CLR	Clears calculator
0001	3 STO A	Stores length of routines
0004	48 STO B	Stores start of routines
0008	DFN F1:ENT@EN	Defines F1
0015	'ROUTINE (1-3)?'	Creates prompt
0029	HLT	Stops program
0030	LBL EN	Labels segment
0033	- 1 =	Adjusts multiplier
0036	* RCL A + RCL B =	Calculates routine address
0043	STO C	Stores routine address
0045	GTO IND C	Executes routine indirectly
0048	GTL AA	Transfers to routine 1
0051	GTL BB	Transfers to routine 2
0054	GTL CC	Transfers to routine 3
0057	LBL AA	Labels segment
0060	'ONE'	Short alpha message
0063	HLT	End of routine 1
0064	LBL BB	Labels segment
0067	1.0045*2.012 =	Longer routine
0080	HLT	End of routine 2
0081	LBL CC	Labels segment
0084	CLR	Very short routine
0085	HLT	End of routine 3

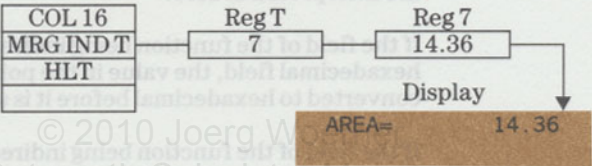
Run the program and try the three options.

Depending upon your applications, you can indirectly address virtually all of the calculator instructions that have fields. Examples of functions that you might want to address indirectly in a program include alpha, test, and file I/O instructions. This section gives examples illustrating an indirect alpha instruction and an indirect test.

Indirect Alpha Merge

By using an indirect alpha merge instruction, you can instruct the calculator to merge the contents of the data register specified by the pointer register.

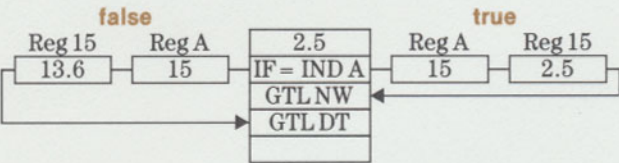
For example, if data register T contains the value 7, the instruction sequence MRG IND T merges the value in data register 007 with the current alpha message. This example is illustrated below.



Indirect Comparison Tests

By using an indirect display-comparison test, you can instruct the calculator to compare the contents of the numeric display register with the contents of the data register specified by the pointer register.

For example, if data register A contains the value 15, the instruction sequence 2.5 IF = IND A compares 2.5 with the contents of data register 15. If the result of the test is true, execution proceeds normally. If the result is false, the first instruction following the comparison test is skipped. This example is illustrated below.



This section defines the **2nd [IND]** function.

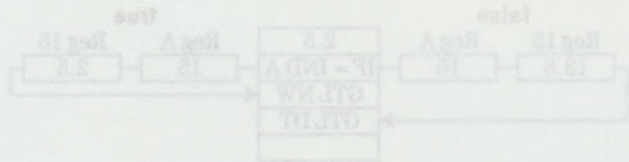
**Indirect** **2nd [IND] nnn** or **X**—Instructs the calculator to use the contents of data register *nnn* or *X* as the field of the function preceding the indirect instruction. This form of addressing is called indirect addressing. Data register *nnn* or *X* is called the pointer register.

If the pointer register contains a positive value with a fractional part, only the integer portion is used as the field. Values less than one, including all negative values, are interpreted as zero.

If the field of the function being indirectly addressed is a hexadecimal field, the value in the pointer register is converted to hexadecimal before it is used.

If the field of the function being indirectly addressed is an alpha field, the characters in the pointer register are read from left to right and only as many characters as are needed for the field are taken from the register. Any extra characters are ignored.

For information on which functions can use indirect addressing, refer to the “Summary of Field Instructions” in Appendix C.



# Chapter 7: Partitioning Memory

---

This chapter discusses the organization of memory in the TI-95 and describes how you can use partitioning to make the most efficient use of available memory.

---

<b>Table of Contents</b>	Introduction .....	7-2
	Why Change Memory Partitions? .....	7-4
	Effects of Partitioning .....	7-6
	Determining the Current Partitions .....	7-8
	Partitioning from the Keyboard .....	7-9
	Partitioning from Within a Program .....	7-12
	Reference Section .....	7-14

© 2010 David Wheeler  
DataArch Calculator Museum

Understanding how the TI-95 memory is organized will help you use the partitioning information in this chapter.

### Divisions of Memory

The TI-95 memory can be thought of as having two main divisions: system memory and user memory.

System memory
User Memory

In typical calculator operations, you do not work directly with system memory.

### Organization of User Memory

User memory consists of 7200 bytes and is divided (partitioned) into file space, program memory, and data registers. Each of the three divisions of user memory has a specific purpose.

File Space
Program Memory
Data Registers

Stores both programs and data in the form of named files. (File operations are discussed in the next chapter.)

Stores programs that you write.

Stores numeric values and alpha messages.

Partitioning lets you increase or decrease an area of user memory, trading one type of memory for another. Some examples are shown here, but there are many other possibilities.

## The Default Partitions

The first time you turn the calculator on, user memory is partitioned as 5200 bytes of file space, 1000 steps (1000 bytes) of program memory, and 125 data registers (1000 bytes).

Although the default partitions are suitable for most situations, you may have applications that require changing the amount of memory allocated for each purpose.

## Actions That Affect the Partitions

When you change the partitions, the newly selected partitions remain in effect and are changed only if you:

- ▶ Repartition the memory.
- ▶ Use **HELP** as a keyboard command to set calculator defaults.
- ▶ Execute **HELP** as a program instruction.
- ▶ Remove the calculator's batteries or allow them to discharge.

After increasing program memory	
file byte 000	file byte 5199
step 0000	step 0000
step 1399	step 1399
register 074	register 000

Default partitions	
file byte 0000	file byte 5199
step 0000	step 0000
step 0000	step 0000
register 124	register 000

# Why Change Memory Partitions?

Partitioning lets you increase or decrease an area of user memory, trading one type of memory for another. Some examples are shown here, but there are many other possibilities.

## Increasing the Data Registers

You might write a program that generates a large amount of data, requiring an additional 50 data registers. Each data register requires eight bytes of memory. The 400 additional bytes are taken from the memory originally partitioned as the highest-numbered program steps (steps 600–999).

### Default partitions

file byte 0000
⋮
file byte 5199
step 0000
⋮
step 0999
register 124
⋮
register 000

### After increasing data registers

file byte 0000
⋮
file byte 5199
step 0000
step 0599
register 174
⋮
register 000

## Increasing the Program Memory

You might want to write a large program that requires more than 1000 steps but uses only a few data registers. Because each data register occupies eight bytes of memory, you can gain 400 program steps by sacrificing only 50 data registers.

### Default partitions

file byte 0000
⋮
file byte 5199
step 0000
⋮
step 0999
register 124
⋮
register 000

### After increasing program memory

file byte 000
⋮
file byte 5199
step 0000
⋮
step 1399
register 074
register 000

## Eliminating File Space

When using an optional Constant Memory™ cartridge for file storage, you may want to partition the calculator's file space to zero. This makes all user memory available for large programs and large amounts of data.

This illustration shows the resulting partitions if you set the file space to zero, the program memory to 4000 steps, and the number of data registers to 400.

### Default partitions

file byte 0000
⋮
file byte 5199
step 0000
⋮
step 0999
register 124
⋮
register 000

### After eliminating file space

step 0000
⋮
step 3999
register 399
⋮
register 000

**Note:** You cannot repartition file space that is occupied by files you have saved. If you attempt to do so, the calculator displays the message **FILES IN USE**. This prevents you from inadvertently destroying files through repartitioning.

An increase in one area of user memory must be accompanied by a decrease in at least one of the other areas. To decide which area to decrease, the calculator treats file space, program memory, and data registers with different priorities. Understanding these priorities helps you understand the effects of repartitioning.

---

### File Space

Increasing the file space reduces the data registers and, if necessary, program steps.

When you increase the file space, the calculator shifts the program toward the data registers, replacing the highest-numbered registers with the highest-numbered steps. This preserves the program and the data in the lower-numbered registers.

If an increase in file space requires more memory than that available in the data registers alone, the additional memory is taken from the highest-numbered program steps. This preserves program instructions located in the lower-numbered steps.

Reducing the file space increases the number of data registers, while preserving the program in memory. To do this, the calculator shifts the contents of program memory so that step 0000 immediately follows the file space. The memory vacated by the shifted program is then filled with zeros and partitioned as additional data registers.

**Note:** You cannot repartition file space that is occupied by files—even by explicitly reducing the file space. If you need occupied file space for program memory or data registers, you must first delete one or more of the files and then reduce the file space.

### Program Memory

When you change the partition for program memory, only the boundary between data registers and program memory is changed. The change does not affect the contents of memory or the partitioning of file space.

---

## **Program Memory (Continued)**

When you increase program steps, the additional steps are taken from the highest-numbered data registers. This preserves data in the lowest-numbered registers, when possible. If you want to increase program steps without decreasing data registers, first decrease the amount of file space and then change the partition for program steps or data registers.

When you reduce program steps, the highest-numbered steps are partitioned as additional data registers. This preserves the lowest-numbered program steps, when possible.

## **Data Registers**

As with a change in program steps, changing the partition for data registers changes only the boundary between data registers and program memory. The change does not affect the contents of memory or the partitioning of file space.

When you increase the number of data registers, the additional registers are taken from the highest-numbered program steps. This preserves the lowest-numbered steps, when possible.

When you reduce data registers, the highest-numbered registers are partitioned as additional program steps. This preserves the lowest-numbered registers, when possible.

## **Clearing Data Registers and Program Memory**

If you clear either the data registers or the program memory and subsequently change the partition for either area, you may find that the increased area is no longer completely cleared. For example, if you clear the data registers (**2nd** **LCMS1**) and then increase the partition for data registers, the calculator interprets previously-stored program instructions as numeric data.

When you want to make sure a new area is cleared, set its partition first and then clear the area.

## Determining the Current Partitions

---

You can determine the current partitions using one of two keyboard commands or a program instruction.


---

### Keyboard Commands

To display the current partitions directly from the keyboard, use either of these two keyboard commands.

- ▶ Press **INV** **2nd** **[PART]**.
- ▶ Press **LIST** **<ST>** to display the current status of the calculator.

The partitions are displayed as shown in this example.



P0400, R100, F6000

In this case, the 7200 bytes of user memory are partitioned as:

- ▶ 400 program steps (400 bytes)
- ▶ 100 data registers (800 bytes)
- ▶ 6000 bytes of file space

### Program Instruction

One characteristic of **INV** **2nd** **[PART]** makes it particularly useful as a program instruction.

It places the number of data registers in the numeric display register and places a number in the t-register that has the form:

*pppp.ffff*

where *pppp* is the number of program steps and *ffff* × 1000 is the number of bytes of file space partitioned.

After executing **INV** **PAR**, your program can examine those registers to determine the partitions.

## Partitioning from the Keyboard

---

You can use keyboard commands to examine possibilities for repartitioning and to set new partitions.

---

### The Partition Menu

As a keyboard command, the key sequence **[2nd] [PART]** displays the partition menu. You use the menu to propose changes in the partitions and to set the changes. (This illustration assumes the default partitions are in effect.)

```
P1000,R125,F5200
PS  REG  FIL  SET  ESC
```

- <PS> Specifies new size for program memory.
- <REG> Specifies new size for data registers.
- <FIL> Specifies new size for file space. You must specify more than 16 bytes of file space; otherwise, the size defaults to zero.
- <SET> Sets the newly-specified sizes.
- <ESC> Cancels the partitioning operation.

Although the sizes of the three areas are displayed as you specify new partitions, the new partitions are not set unless you press <SET>. This characteristic of the partition menu lets you examine possibilities for partitioning without making any changes.

**Note:** If the program steps or file space size you specify is not a multiple of eight bytes, the size is rounded upward to the next multiple of eight.

(continued)

**Procedure** To set partitions directly from the keyboard:

1. Press **[2nd][PART]** to display the partition menu.
2. Indicate the area of memory you want to change by pressing **<PS>**, **<REG>**, or **<FIL>**.

The calculator displays a prompt for you to propose a new size for the area.

For example, pressing **<PS>** displays:



ENTER PROG STEPS  
ENT ESC

3. Enter the proposed size for the area you selected in either steps of program memory, number of data registers, or number of bytes of file space. (Pressing **<ESC>** at this point returns you to the first partitions menu.)
4. Press **<ENT>** to enter the number.

The calculator displays one of the following.

- ▶ The new partitions that will be used if you set them.
- ▶ An error message indicating you have specified more memory than is available for that area. To clear the error condition and try again, press **[CLEAR]**.

### Procedure (Continued)

5. When you have specified the memory partitions you want to change, press <SET> to set the partitions. The calculator then:

- ▶ Sets and displays the partitions.
- ▶ Places the number of data registers in the numeric display register.
- ▶ Places a number in the t-register with the form *pppp.ffff* where *pppp* is the number of program steps and *ffff* is the number of bytes of file space.

### Example

The following example, entered directly from the keyboard, changes from the default partitions to 6000 bytes of file space, 400 program steps, and 100 data registers. The example then restores the default partitions.

Procedure	Press	Display
Select partition menu	<b>2nd</b> [PART]	P1000,R125,F5200
Specify 6000 bytes	<FIL> 6000 <ENT>	P1000,R025,F6000
Specify 400 steps	<PS> 400 <ENT>	P0400,R100,F6000
Set new partitions	<SET>	P0400,R100,F6000
Select partition menu	<b>2nd</b> [PART]	P0400,R100,F6000
Specify 5200 bytes	<FIL> 5200 <ENT>	P0400,R200,F5200
Specify 125 registers	<REG> 125 <ENT>	P1000,R125,F5200
Set partitions	<SET>	P1000,R125,F5200

## Partitioning from Within a Program

---

You can include instructions in your program to partition memory. This makes it unnecessary to set partitions from the keyboard before running a program. When you set partitions in a program, you must provide all the necessary information to the partition function; the function does not prompt you for the information.

---

**Procedure** To set partitions from within a program:

1. Determine the amount of memory you want to allocate for each of the three areas of user memory.
2. Have the program place a value in the numeric display register that has the form:

*pppp.ffff*

where *pppp* is the number of program steps and *ffff* is the number of bytes of file space you want. (You must include any leading zeros in *ffff*.)

- ▶ If you do not want to change the size of the file space, make *ffff* equal to 0.
  - ▶ If you want to set the file space to 0 bytes, make *ffff* in the range 0001 through 0016.
3. Have the program execute the PAR instruction.
  4. When the sequence *pppp.ffff* PAR is executed, the calculator:
    - ▶ Sets the program memory and file space to the sizes you specified and allocates the remaining user memory to the data registers.
    - ▶ Places the resulting number of data registers in the numeric display register.
    - ▶ Places the resulting number of program steps and file bytes (in the form *pppp.ffff*) in the t-register.

Use this section as a source of reference information for partitioning memory.

**Example** The following program segment partitions user memory to the default settings.

PC =	Program Mnemonics	Comments
0000	1000.5200	1000 program steps, 5200 bytes file space
0009	PAR	Sets the partitions
	⋮	
	(Remainder of program)	
	⋮	

© 2010 Joerg Woerner

Datamath Calculator Museum

Use this section as a source of reference information for partitioning memory.

---

### Determine Current Partitions

**[INV] [2nd] [PART]**—Displays the current partition settings for program memory, data registers, and file space. The function also places the number of data registers in the numeric display register and places the number of program steps (*pppp*) and number of bytes of file space (*ffff*) in the t-register in the form *pppp,ffff*.

**[INV] [2nd] [PART]** can be used as a keyboard command or as a program instruction. The instruction mnemonics are INV PAR.

### Set Partitions: Keyboard Command

**[2nd] [PART]**—As a keyboard command, displays the partitions menu and shows the current partitions. The menu is used to propose new partitions and to set the proposed partitions.

**D <PS> *nnnn* <ENT>**—Proposes *nnnn* as the number of program steps. If *nnnn* is not a multiple of eight bytes, the calculator adjusts it upward to the nearest multiple of eight. The calculator determines any resulting change in the number of data registers. (File space is never affected by partitioning program memory.) It then redisplayes the partitions menu to show the effect that the proposed change will have. The partitions are not changed unless you press <SET>.

**<REG> *nnn* <ENT>**—Proposes *nnn* as the number of data registers. The calculator determines any resulting change in the number of program steps. (File space is never affected by partitioning data registers.) It then redisplayes the partitions menu to show the effect that the proposed change will have. The partitions are not changed unless you press <SET>.

---

**Set Partitions:** `<FIL> nnnn <ENT>`—Proposes *nnnn* as the partition for file space. If *nnnn* is not a multiple of eight bytes, the calculator adjusts it upward to the nearest multiple of eight. The maximum size you can partition for file space is 6200 bytes. The calculator determines any resulting change in the number of data registers and, if necessary, program memory. It then redisplay the partitions menu to show the effect that the proposed change will have. The partitions are not changed unless you press `<SET>`.

**Note:** If *nnnn* is less than that required for files you have saved, the calculator displays an error message. Because a minimum of 16 bytes are required for the catalog of a directory, the calculator displays 0 as the proposed partition for file space if *nnnn* is 16 or less.

`<SET>`—Sets the calculator's partitions to those displayed in the partitions menu and exits the menu. `<SET>` also places the number of data registers in the numeric display register and the number of program steps (*pppp*) and bytes of file space (*ffff*) in the t-register in the form *pppp,ffff*.

`<ESC>`—Exits the menu without changing the settings of any partitions.

(continued)

### Set Partitions: Program Instruction

**[2nd] [PART]**—In a program, sets the number of program steps and amount of file space according to the number in the numeric display register. No menu is displayed.

The number must be in the form *pppp.ffff*, where *pppp* is the number of program steps and *ffff* is the number of bytes of file space. You must include any leading zeros in *ffff*. If a requested partition is not a multiple of eight bytes, the calculator adjusts it upward to the nearest multiple of eight. The maximum size you can partition for file space is 6200 bytes.

If *ffff* is zero, the partition for file space is not changed. Because a minimum of 16 bytes are required for the catalog of a directory, the file space is set to zero if *ffff* is in the range 1 through 16. If *ffff* is less than the amount required for the files you have saved already, the calculator displays an error message.

After setting the partitions for program steps and file space, the calculator partitions any remaining memory as data registers.

The function places the number of data registers in the numeric display register and places the number of program steps (*pppp*) and bytes of file space (*ffff*) in the t-register in the form *pppp.ffff*.

The instruction mnemonic for **[2nd] [PART]** is PAR.

## Chapter 8: File Operations

---

This chapter describes how to use the TI-95 file space or an 8K Constant Memory™ cartridge for saving and retrieving your programs and data. (If you are using a cassette tape for storage, refer to Chapter 9 for instructions.)

---

### Table of Contents

Introduction .....	8-2
Using the <b>FILE STORAGE</b> Menu .....	8-4
If Using a Constant Memory Cartridge .....	8-5
Saving a Program as a File .....	8-6
Running a Program File .....	8-8
Loading a Program File .....	8-10
Saving Data as a File .....	8-12
Loading a Data File .....	8-16
Listing the Catalog of a Directory .....	8-18
Deleting Files .....	8-20
Performing File Operations in a Program .....	8-22
Reference Section .....	8-24

© 2010 Joerg Woelke  
DataMath Calculator Museum

By storing programs and data as files, you can reuse them without the need to enter the keystrokes again. While the information is in a file, you can use the calculator's program memory and data registers for other purposes.

## Types of Files

You can create two types of files: program files and data files.

A program file is a copy you make of the contents of program memory.

- ▶ You can run a program while it is stored as a file. You do not have to first load it into program memory.
- ▶ You cannot list or modify a program unless you first load it back into program memory.

A data file is a copy you make of the contents of a series of data registers.

- ▶ Data you may need to reexamine or update can be saved as a data file.
- ▶ You cannot list, modify, or otherwise use the contents of a data file unless you first load it back into data registers.

## Rules for Naming Files

When you create a file, you specify a three-character name for it. The name can include letters, numerals, and punctuation symbols. If you use fewer than three characters, the calculator adds trailing spaces. (You do not activate the alpha mode to fill in this field.)

You must identify a data file by using a + as the first character of the file name (for example, +SQ). You cannot use + as the first character for a program file. This rule lets the calculator distinguish between a program file and a data file.

If you make a mistake while entering a file name in a keyboard command, you must use the **CLEAR** key to erase the name and start over.

---

## File Directories

Because you can save a file in either a Constant Memory cartridge or in the calculator's file space, you need a way to specify which storage area you want to use.

The calculator lets you do this by treating each file-storage area as a named directory. While using the menus described in this chapter, keep in mind:

- ▶ The directory name **MEM** always refers to the calculator's file space.
- ▶ The directory name **NEW** is assigned by the calculator to any Constant Memory cartridge when you first use the cartridge. (You can change the name to one you prefer.)

## Types of File Operations

The TI-95 lets you perform several file operations. These include:

- ▶ Saving a program or data as a file.
- ▶ Loading a program or data back into memory.
- ▶ Viewing a catalog of files saved in a specified directory.
- ▶ Deleting a specified file.
- ▶ Clearing all the files in a specified directory.
- ▶ Renaming a Constant Memory cartridge.

You can perform file operations directly from the keyboard, or from within a program. Both methods are described, but you should try the keyboard examples before using file operations in your programs.

## Using the FILE STORAGE Menu

---

The **[FILES]** key presents a menu to let you perform various types of file operations, such as saving, loading and deleting files.

---

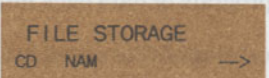
### The FILE STORAGE Menu

When you press **[FILES]**, the following menu is displayed. You use this menu whether working with file space or a Constant Memory cartridge.



```
FILE STORAGE
GET  PUT  DF  CAT  -->
```

- <GET> Loads a file you have saved back into program memory or data registers.
- <PUT> Saves a program file or a data file.
- <DF> Deletes a specified file from the selected directory.
- <CAT> Displays a catalog of file names in the selected directory.
- <-->> Displays additional selections (shown below).



```
FILE STORAGE
CD   NAM   -->
```

- <CD> Clears all files in a specified directory.
- <NAM> Lets you assign a name to a Constant Memory cartridge.
- <-->> Displays previous selections (shown above).

Read the information on this page if you are using a Constant Memory cartridge and you have not set the partition for the file space to zero.

### Using the Menu with a Cartridge

If you are using a Constant Memory cartridge and you have partitioned file space, when you press <GET>, <PUT>, <DF>, or <CAT>, an additional menu lets you specify which directory you want to use.

For example, when you press <PUT>, the calculator displays:



<MEM>      Selects the file space.

<NEW>      Selects the Constant Memory cartridge. (The name NEW is automatically assigned to a new cartridge, but can be changed, as shown below.)

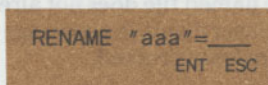
### Renaming the Cartridge

The <NAM> selection of the **FILE STORAGE** menu lets you rename a cartridge by assigning a three-character name of your choice.

To rename a cartridge:

1. Press <--> <NAM> from the **FILE STORAGE** menu.

The calculator displays:



(where *aaa* is the name of the cartridge)

2. Enter a three-character name you want the cartridge to have and press <ENT> (or press <ESC> to cancel the operation). To correct a mistake while entering the name, you must press **CLEAR**.

The calculator renames the cartridge and displays the message **DIR RENAMED**.

When you create a program file, the entire program is saved, starting with the instruction at program step 0000 and ending with the last instruction in the program. (Because a program is saved in multiples of eight bytes, the calculator might save a few NOP instructions following the program.)

**Procedure** To save the program contained in program memory:

1. Press **[FILES]** to display the **FILE STORAGE** menu.
2. Press **<PUT>**. (If a Constant Memory cartridge is installed and you have file space partitioned, you must then select either the cartridge or the file space as the current directory.)

The calculator displays:



SAVE PGM OR REGS  
PGM REG ESC

3. Press **<PGM>**.

The calculator displays the following. (If you selected the cartridge as the current directory, the name of the cartridge is shown instead of MEM.)



DIR=MEM, FILE=\_\_\_\_\_  
ENT ESC

4. Enter a three-character name for the program file and press **<ENT>**. Do not use **+** as the first character.
  - ▶ If the named file does not already exist, the calculator displays a message to confirm that the file has been created.
  - ▶ If the current directory already contains a file with the name you have specified, the calculator displays the menu shown on the next page.

### Procedure (Continued)

When the named file already exists, the calculator displays:

REPLACE aaa?  
YES NO

(where *aaa* is the name of the file)

- ▶ To replace the existing file with the current contents of program memory, press <YES>. The new file does not have to be the same size as the existing file.
- ▶ To cancel the save operation and leave the existing file unaltered, press <NO>. The current contents of program memory are not saved.

### Example

- So that you will have a program example to save, enter the short program shown on page 1-11 of this guide. Then make sure you have file space partitioned and use the following keystrokes to save the program as a file named **SPL** in the file space.

Procedure	Press	Display
Save file*	<b>FILES</b> <PUT>	SAVE PGM OR REGS
Specify the file is a program	<PGM>	DIR = MEM, FILE = ____
Name the file <b>SPL</b>	<b>SPL</b> <ENT>	FILE SPL SAVED

\*If you have a Constant Memory cartridge installed, you must select <MEM> as the current directory after pressing <PUT>.

## Running a Program File

You can run a program you have saved as a file without first loading it into program memory. This feature lets you build a library of programs that you can easily access.

### Limitations on Running a File

Steps in program files are numbered starting with 0000, just as they are in program memory. This lets the program use transfer instructions with absolute addressing as well as label addressing.

When you use **RUN** as a keyboard command to run a program file, execution always begins at step 0000. You cannot use the keyboard commands SBL and SBR to start at a different location in the file; these keyboard commands always refer to the program in memory.

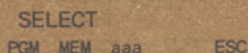
You can, however, use GTL, GTO, SBL, and SBR as program instructions to execute specific routines in a file. The reference section at the end of this chapter describes how to do this.

### Procedure

To run a program as a file:

1. Press **RUN**.

The calculator displays:



SELECT  
PGM MEM aaa ESC

(where *aaa* is the name of an optional Constant Memory cartridge)

**Note:** Your display may not show all the menu selections shown here. For example, if you have the calculator's file space partitioned as zero bytes, the MEM selection does not appear.

## Procedure (Continued)

2. Select the directory that contains the program.

The calculator displays:

```
SELECT FILE:
aaa  bbb  ccc  -->
```

(where *aaa*, *bbb*, and *ccc* are names of program files in the selected directory)

3. If your program requires you to enter a number into the display before running the program, enter the number.
4. Select the program you want to run. (If necessary, press <--> until you see the name of the program.)

The calculator runs the program, beginning at program step 0000.

## Example

Run the sample program file that you saved under the name SPL. (This example assumes you entered and saved the program from page 1-11 as directed by the example on page 8-7. If you have saved other program files, the names of those files will also be listed on the <MEM> menu.)

Procedure	Press	Display
Begin	<b>RUN</b>	SELECT: PGM MEM      ESC
Select the file space	<MEM>	SELECT FILE: SPL          -->
Enter a value	5	5
Calculate $5^3$	<SPL>	125.

## Loading a Program File

---

Although you can run a program without loading it into program memory, you might want to load the program to list or edit it. When you load a program file, the entire program memory is cleared, and the file is copied into program memory starting at program step 0000.

---

**Procedure** To load a program file into program memory:

1. Make sure program memory does not contain a program you may need later.
2. Press **FILES** to display the **FILE STORAGE** menu.
3. Press **<GET>**. (If a Constant Memory cartridge is installed and you have file space partitioned, you must then select either the cartridge or the file space as the current directory.)

The calculator displays:



LOAD PGM OR REGS  
PGM REG                      ESC

4. Press **<PGM>**.

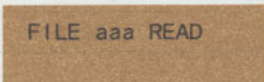
The calculator displays the following. (If you selected the cartridge as the current directory, the name of the cartridge is shown instead of MEM.)



DIR=MEM, FILE=\_\_\_\_  
                                  ENT  ESC

5. Enter the same three-character name you used to save the program file and press **<ENT>**.

The calculator displays:



FILE aaa READ

(where *aaa* is the name of the file)

When you create a data file, you specify both the number of data registers to be saved and the starting register of the series that contains the data.

### Example

Use the following sequence of keystrokes to load the program you saved back into program memory.

Procedure	Press	Display
Load a file	<b>FILES</b> <GET>	LOAD PGM OR REGS
Specify that the file is a program	<PGM>	DIR = MEM, FILE = ____
File name is SPL	SPL <ENT>	FILE SPL READ

After loading the program file, you can use the learn mode to confirm it has been loaded into program memory.

Datamath Calculator Museum



## Saving Data as a File

---

When you create a data file, you specify both the number of data registers to be saved and the starting register of the series that contains the data.

---

### Procedure

To save a series of data registers as a file:

1. Press **FILES** to display the **FILE STORAGE** menu.
2. Press **<PUT>**. (If a Constant Memory cartridge is installed and you have file space partitioned, you must then select either the cartridge or the file space as the directory.)

The calculator displays:



SAVE PGM OR REGS  
PGM REG                      ESC

3. Press **<REG>**.

The calculator displays:



ENTER # OF REGS  
ENT ESC

4. Specify both the number of registers to be saved and the address of the first register by entering a number. The number must be in the form:

*nnn.sss*

where *nnn* specifies the number of registers, and *sss* (including any leading zeros) specifies the starting register. If you do not include *sss*, the calculator assumes the starting register to be 000.

5. Press **<ENT>** to enter the number.

The calculator displays the menu shown on the next page.

**Procedure  
(Continued)**

When you press <ENT>, the calculator displays the following. (If you selected the cartridge as the current directory, the name of the cartridge is shown instead of MEM.)



DIR=MEM, FILE=+  
ENT ESC

6. Enter two characters following the + as a name for the data file and press <ENT>. (The calculator supplies the required + as the first character.)
  - ▶ If the named file does not already exist, the calculator displays a message to confirm that the file has been created.
  - ▶ If the named file already exists, the calculator displays the menu shown on the next page.

(continued)

### Procedure (Continued)

If the current directory already contains a data file with the name you have specified, the calculator displays a menu to let you either replace the existing file or cancel the save operation.



REPLACE +nn?  
YES NO

(where +nn is the name of the file)

- ▶ To replace the existing file, press <YES>. The new file does not have to be the same size as the existing file.
- ▶ To cancel the save operation and leave the existing file unaltered, press <NO>. The data registers are not saved.

Datamath Calculator Museum

### Example

Make sure you have file space partitioned and then use the following sequence to store three values in data registers 010, 011, and 012 and save them as a data file named +SP in the file space.

Procedure	Press	Display
Clear data registers	<b>2nd</b> <b>[CMS]</b>	0.
Store three values	10 <b>[STO]</b> 010 11 <b>[STO]</b> 011 12 <b>[STO]</b> 012	10. 11. 12.
Save registers*	<b>[FILES]</b> <PUT> <REG>	ENTER # OF REGS
3 registers, starting with register 010	3.010 <ENT>	DIR = MEM, FILE = +__
Name the file + SP	SP <ENT>	FILE + SP SAVED

\*If you have a Constant Memory cartridge installed, you must select <MEM> as the current directory after pressing <PUT>.

## Loading a Data File

When loading a data file, you specify the starting register into which the data is to be loaded—usually the same register from which the data was saved. The entire data file is loaded into memory, replacing the previous contents of the registers.

### Procedure

To load a data file into data registers:

1. Make sure the data registers to be filled do not contain data you may need later.
2. Press **[FILES]** to display the **FILE STORAGE** menu.
3. Press **<GET>**. (If a Constant Memory cartridge is installed and you have file space partitioned, you must then select either the cartridge or the file space as the directory.)

The calculator displays:

```
LOAD PGM OR REGS
PGM  REG          ESC
```

4. Press **<REG>**.

The calculator displays:

```
ENTER 1st REG #
          ENT  ESC
```

5. Enter the lowest of the registers into which data is to be loaded and press **<ENT>**.

The calculator displays the following. (If you selected the cartridge as the current directory, the name of the cartridge is shown instead of MEM.)

```
DIR=MEM, FILE=+___
          ENT  ESC
```

### Procedure (Continued)

6. Enter the two characters you specified following the + when you saved the data file and press <ENT>.

The calculator displays a message confirming that the data has been read.

### Example

Use the following sequence of keystrokes to load the three data values you saved into registers 020, 021, and 022. This example shows that you can load the data into a different series of registers than those from which it was saved.

Procedure	Press	Display
Clear display and data registers	<b>CLEAR</b> <b>2nd</b> <b>[CMS]</b>	0.
Load a data file*	<b>FILES</b> <GET> <REG>	ENTER 1st REG #
Specify register 20	20 <ENT>	DIR = MEM, FILE = +__
Specify file name + SP	SP <ENT>	FILE + SP READ
Confirm load was OK	<b>RCL</b> 020	10.
	<b>RCL</b> 021	11.
	<b>RCL</b> 022	12.

\*If you have a Constant Memory cartridge installed, you must select <MEM> as the current directory after pressing <GET>.

## Listing the Catalog of a Directory

---

The <CAT> (catalog) selection of the FILE STORAGE menu lets you examine the name and size of each file in a specified directory. This selection also lists the amount of available storage space remaining. If a printer is attached and operational, the list is automatically printed.

---

**Procedure** To list the files saved in a directory:

1. Press **[FILES]** to display the **FILE STORAGE** menu.
2. Press <CAT>. (If a Constant Memory cartridge is installed and you have file space partitioned, you must then select either the cartridge or the file space as the current directory.)

The calculator displays the following.

- The name of the directory.
- The name of each file and the file's size in bytes.
- The number of bytes free (available for saving additional files). The calculator also places this number in the numeric display register.

If you do not have a printer attached, the calculator normally pauses for one second before displaying each item.

However, you can use the **[→]** key to control the speed of the listing.

- To pause the listing indefinitely, hold down the **[→]** key.
- To advance to the next group of file names without the one-second pause, press and release the **[→]** key.

**Example** Use the following sequence of keystrokes to list all files currently saved in your directory.

Procedure	Press	Display
Select menu	<b>FILES</b>	FILE STORAGE
Select file catalog*	<b>&lt;CAT&gt;</b>	DIRECTORY = MEM
Sample program file		SPL: 8 BYTES
Sample data file		+ SP: 24 BYTES
Available space		BYTES FREE: 5144

\*If you have a Constant Memory cartridge installed, you must select **<MEM>** as the current directory after pressing **<CAT>**.

If you have saved other files in addition to the sample program and data files, the additional files are included in the listing. The free space shown by your calculator may vary from the example, depending upon the size of the file space and the number of files you have saved.

## Deleting Files

You can either delete a specified file or clear all files in a specified directory.

### CAUTION

You cannot recover files after deleting them. Before you delete a file or clear a directory, make sure you no longer need the information.

### Deleting a Specified File

To delete a specified program file or data file:

1. Make sure you no longer need the file.
2. Press **FILES** to display the **FILE STORAGE** menu.
3. Press <DF> (delete file). (If a Constant Memory cartridge is installed and you have file space partitioned, you must then select either the cartridge or the file space as the directory.)



DIR=MEM, FILE=\_\_\_  
ENT ESC

4. Enter the name of the file you want to delete and press <ENT>. If the file is a data file, you must enter + as the first character.

The calculator displays a message to confirm the file has been deleted.

### Example

Use the following sequence of keystrokes to delete the data file you saved.

Procedure	Press	Display
Select delete file	<b>FILES</b> <DF>	DIR = MEM, FILE = ___
Enter the file name	<b>+</b> SP <ENT>	FILE + SP DELETED

## Clearing a Directory

To clear all files in a specified directory:

1. Make sure you no longer need any files in the directory you are clearing.
2. Press **[FILES]** to display the **FILE STORAGE** menu.
3. Press <--> <CD> (clear directory).

The calculator displays:

CLEAR DIRECTORY  
MEM aaa ESC

(where *aaa* is the name of the cartridge, if installed)

4. Select either the name of the directory you want to clear or <ESC>.
  - If you select the name of a directory, the calculator displays a message to confirm that the directory has been cleared.
  - If you select <ESC>, the directory is not cleared.

## Example

Use the following key sequence to clear the calculator's file space of all files.

Procedure	Press	Display
Select clear directory*	<b>[FILES]</b> <--> <CD>	CLEAR DIRECTORY MEM ESC
Specify file space	<MEM>	DIR CLEARED

\*If you have a Constant Memory cartridge installed, the calculator displays the name of the cartridge in addition to MEM.

## Performing File Operations in a Program

---

You can design a program so that it performs its own file operations. After you are familiar with using the file operations from the keyboard, read this section to learn how to use them within a program.

---

### Important Concept

The key sequence for entering file operations into a program is different from that used in keyboard commands. In the learn mode, the calculator prompts you only for the type of file operation and does not display menus for other information, such as the file name.

Within a program you must specify:

- ▶ Which data registers are to be used (if the file is a data file).
- ▶ Which directory is to be used.
- ▶ The name of the file, including the required + sign if the file is a data file.

### Specifying the Data Registers

For a program that saves data, you must ensure the value *nnn.sss* (as described on page 8–12) is in the numeric display register before executing <PUT>.

For a program that loads data, you must ensure the number of the starting register is in the numeric display register before executing <GET>.

### Specifying the Directory

For a program that performs a file operation using a Constant Memory cartridge, you must use the **INV** key immediately preceding the function to specify the cartridge as the directory to be used. The only exception to this is the <NAM> function, which always accesses the cartridge and does not require **INV**.

To specify the cartridge as the directory, press **INV** before you select any of the file operations <PUT>, <GET>, <DF>, <CD>, or <CAT>.

If you do not use **INV**, the calculator uses the file space (MEM) as the directory.

### Specifying the Name and Type of File

For any file operation that requires a file name, such as <PUT> and <GET>, you must enter the file name immediately following the operation. If you use fewer than three characters, the calculator adds trailing spaces. (You do not activate the alpha mode to enter the name; the calculator interprets the keys as alpha characters.)

If the file is a data file, you must include the required + as the first character of the file name (for example, +SQ).

### Sample Key Sequences

The following samples show some key sequences typical of those you might enter in the learn mode for performing file operations from a program.

**FILES** 10.002  
**INV** <PUT> +GP

Saves 10 registers (starting with register 002) as a data file named +GP in the cartridge.

**FILES** 5 <GET> +CA

Loads the data file named +CA from the file space into data registers, starting at register 005.

**FILES** **INV** <DF> ABC

Deletes the program file named ABC from the cartridge.

Use this section as a source of reference information for file operations.

---

### File Operations

**[FILES]**—Displays a menu of file operations. The menu lets you save (<PUT>) programs and data as named files in one of two directories: the calculator's file space or an installed Constant Memory cartridge. You also use the menu to load (<GET>) a file you have saved, show a catalog of files in a directory (<CAT>), delete a specified file (<DF>), clear all files from a directory (<CD>), or name a Constant Memory cartridge (<NAM>).

Before carrying out a save or load operation, the calculator checks the destination to make sure enough room exists for the file. If insufficient room exists, the save or load is not performed, and the destination is not altered. When you replace an existing file with a new file of the same name, the new file need not be the same size as the old file. The calculator repositions other files to accomodate any difference in size.

### Program Files

When you save a program as a file, you assign a three-character file name. The first character must not be a + because that character identifies a data file. The entire program is saved, starting with the instruction at program step 0000 and ending with the last instruction in the program. (Because a program file is saved in multiples of eight bytes, the calculator might include a few NOP instructions at the end of the program.) When you load a program file into program memory, the file is loaded starting at program address 0000.

### Data Files

When you save a data file, you assign a three-character file name. The first character of the name must be a +. This gives the calculator a way to identify the file as a data file. You specify the starting data register and the number of registers to save. When you load a data file, the entire file is loaded into data registers, beginning at the register you specify. You can load a data file into a different series of registers than those from which the data was saved.

---

**File Directories** The calculator treats each file-storage area (either a Constant Memory cartridge or the calculator's file space) as a named directory.

The directory name **MEM** refers to the calculator's file space. The name **NEW** is assigned by the calculator to any Constant Memory cartridge when you first use the cartridge. You can assign a different name to a cartridge. If you use <CD> (clear directory) to clear all files in the cartridge, the calculator reassigns the name **NEW** to the cartridge.

**Keyboard Commands** When you use the **FILE STORAGE** menu from the keyboard, you are prompted for information such as a file name or the number of a starting register.

If you specify fewer than three characters for a file name or directory name, the calculator supplies trailing spaces. If you make a mistake while entering such information, you must press **[CLEAR]** to restart the entry.

**Program Instructions** If you press **[FILES]** while the calculator is in the learn mode, the calculator displays the menu but does not prompt for file names, directory names, or numeric information such as a starting register. Instead, you must supply the required information as either an instruction field or a number in the numeric display register. When you execute the function as a program instruction, the function uses the information you stored.

If you specify fewer than three characters for a file name or directory name, the calculator supplies trailing spaces.

(continued)

**Save  
Program File:  
Keyboard  
Command**

**[FILES] <PUT> <PGM> *aaa* <ENT>**—Saves the program contained in program memory as a file named *aaa* in the current directory. If a Constant Memory cartridge is installed and file space is partitioned, you are prompted to select either the cartridge or the file space as the current directory. The first character of file name *aaa* must not be a + because that character identifies a data file.

If file *aaa* already exists, the calculator prompts you to choose whether you want to replace the file or cancel the file save function.

**Save  
Data File:  
Keyboard  
Command**

**[FILES] <PUT> <REG> *nnn.sss* <ENT> +*aa* <ENT>**—Saves a series of data registers as a file named +*aa* in the current directory. If a Constant Memory cartridge is installed and file space is partitioned, you are prompted to select either the cartridge or the file space as the current directory. The calculator supplies the required + as the first character of file name +*aa*. The number *nnn.sss* specifies the number of registers to be saved (*nnn*) and the address of the starting register (*sss*). You must include any leading zeros in *sss*. If you omit *sss*, the calculator uses 000 as the starting register.

If file +*aa* already exists, the calculator prompts you to choose whether you want to replace the file or cancel the file save function.

(continued)

- 
- Load Program File:**  
**Keyboard Command**     **FILES** <GET> <PGM> *aaa* <ENT>—Clears any program currently contained in program memory and loads program file *aaa* from the current directory into program memory. If a Constant Memory cartridge is installed and file space is partitioned, you are prompted to select either the cartridge or the file space as the current directory. The first character in the file name must not be a +.
- Load Data File:**  
**Keyboard Command**     **FILES** <GET> <REG> *sss* <ENT> + *aa* <ENT>—Loads the data file named + *aa* from the current directory into a series of data registers. If a Constant Memory cartridge is installed and file space is partitioned, you are prompted to select either the cartridge or the file space as the current directory. The file data is loaded starting at register *sss* and replaces any data already in the destination registers. Register *sss* need not be the same register from which the data was saved. The calculator supplies the required + as the first character of + *aa*.
- Name Cartridge:**  
**Keyboard Command**     **FILES** <NAM> *aaa* <ENT>—Assigns the name *aaa* to the currently installed Constant Memory cartridge. You should not use the names MEM and PGM because these names refer to the file space and program memory.

(continued)

**Show Directory  
Catalog:  
Keyboard  
Command**

**[FILES]** <CAT>—Displays the name of the current directory, the name of each file in the directory, the file's size in bytes, and the total number of bytes free in the directory. If a Constant Memory cartridge is installed and file space is partitioned, you are prompted to select the cartridge or the file space as the current directory. The calculator places the number of bytes of free space in the numeric display register.

To stop the listing before it finishes, press the **[BREAK]** or **[HALT]** key.

If a printer is attached, the list is automatically printed. Otherwise, the calculator pauses one second after displaying each item. Holding down **[→]** pauses the listing indefinitely. Pressing and releasing **[→]** advances immediately to the next item.

**Delete File:  
Keyboard  
Command**

**[FILES]** <DF> *aaa* <ENT>—Deletes the data or program file named *aaa* from the current directory. If a Constant Memory cartridge is installed and file space is partitioned, you are prompted to select either the cartridge or the file space as the current directory. If the file is a data file, you must specify + as the first character of the file name.

You cannot recover a file after deleting it.

**Clear  
Directory:  
Keyboard  
Command**

**[FILES]** <CD> <Directory>—Clears the directory named <Directory> by deleting all the files in that directory. The directory can be either the calculator's file space (<MEM>) or a Constant Memory cartridge. (The name of the directory appears as a menu selection.) When you use <CD> with a cartridge, the calculator renames the cartridge as **NEW**.

You cannot recover files from a cleared directory.

---

**Save  
Program File:  
Program  
Instruction**

**FILES** <PUT> *aaa*

or

**FILES** **INV** <PUT> *aaa*—Saves the program contained in program memory as a file named *aaa*. <PUT> (without **INV**) saves the file in the calculator's file space. **INV** <PUT> saves the file in the installed Constant Memory cartridge. The first character of *aaa* must not be a + because that character identifies a data file.

If file *aaa* already exists, the calculator replaces the file.

**Save  
Data File:  
Program  
Instruction**

**FILES** <PUT> + *aa*

or

**FILES** **INV** <PUT> + *aa*—Saves a series of data registers as a file named + *aa*. <PUT> (without **INV**) saves the file in the calculator's file space. **INV** <PUT> saves the file in the currently installed Constant Memory cartridge. You must use + as the first character of the file name. When the function is executed, the numeric display register must contain a number in the form *nnn.sss*, specifying the number of registers to be saved (*nnn*) and the address of the starting register (*sss*). You must include any leading zeros in *sss*. If you omit *sss*, the calculator uses 000 as the starting register.

If file + *aa* already exists, the calculator replaces the file.

(continued)

**Load Program File:** **Instruction** **FILES** <GET> *aaa*  
or  
**FILES** **INV** <GET> *aaa*—Clears the program currently contained in program memory and loads program file *aaa* into program memory. <GET> (without **INV**) loads the file from the calculator's file space. **INV** <GET> loads the file from the installed Constant Memory cartridge. The first character in the file name must not be a +.

**Load Data File:** **Instruction** **FILES** <GET> + *aa*  
or  
**FILES** **INV** <GET> + *aa*—Loads the data file named + *aa* into a series of data registers. <GET> (without **INV**) loads the data from the calculator's file space. **INV** <GET> loads the data from the installed Constant Memory cartridge. You must use + as the first character of the file name. When the function is executed, the numeric display register must contain a number in the form *sss* specifying the starting register. The file data replaces any data already in the registers. Register *sss* need not be the same register from which the data was saved.

**Name Cartridge:** **FILES** <NAM> *aaa*—Assigns the name *aaa* to the currently installed Constant Memory cartridge.

**Program Instruction** The calculator assigns the name **NEW** to a Constant Memory cartridge until you assign another name.

**Show Directory** **FILES** <CAT>

**Catalog:** or

**Program Instruction** **FILES INV** <CAT>—Displays the catalog of the specified directory, including the name of the directory, the name and size of each file in the directory, and the total number of bytes free in the directory. <CAT> (without **INV**) specifies the calculator's file space as the directory. **INV** <CAT> specifies the installed Constant Memory cartridge as the directory. The calculator places the number of bytes of free space in the numeric display register.

If a printer is attached, the list is automatically printed. If a printer is not attached, the calculator normally pauses one second after displaying each item. In this case, holding down **→** pauses the listing indefinitely. Pressing and releasing **→** advances immediately to the next item.

(continued)

**Delete File:  
Program  
Instruction**

**FILES** <DF> *aaa*  
or

**FILES** **INV** <DF> *aaa*—Deletes data or program file *aaa*.  
<DF> (without **INV**) deletes the file from the calculator's  
file space. **INV** <DF> deletes the file from the installed  
Constant Memory cartridge. If the file is a data file, you  
must specify + as the first character of the file name.

You cannot recover a file after deleting it.

**Clear  
Directory:  
Program  
Instruction**

**FILES** <CD>  
or

**FILES** **INV** <CD>—Clears all files from the specified  
directory. <CD> (without **INV**) clears files from the  
calculator's file space. **INV** <CD> clears files from the  
installed Constant Memory cartridge and renames the  
cartridge as **NEW**.

You cannot recover files from a directory you have  
cleared.

(continued)

---

**Run  
Program File:  
Keyboard  
Command**

**RUN** <MEM> <File Name>

or

**RUN** <Directory> <File Name>—Begins execution of the program named <File Name> in either the calculator's file space (<MEM>) or the installed Constant Memory cartridge named <Directory>. (The actual directory and file name appear as menu selections.) The program runs directly from the file space or from the cartridge; it is not loaded into program memory.

**Note:** If the program expects a number to be in the numeric display register, you can enter the number anytime before selecting <File Name>.

(continued)

© 2010 Joerg Woerner

Datamath Calculator Museum

### Execute

### Routine:

### Program

### Instruction

**RUN** **2nd** **[GTL]** *aa*

or

**RUN** **2nd** **[SBL]** *aa*

or

**RUN** **INV** **2nd** **[GTL]** *nnnn*

or

**RUN** **INV** **2nd** **[SBL]** *nnnn*—Transfers control to a routine or calls a subroutine in either the calculator's file space, the installed Constant Memory cartridge, or program memory. This instruction lets your program execute routines that are not located in the same area in which your program is running. A program running from the file space, for example, can call a subroutine in a cartridge or in program memory. You cannot use this function as a keyboard command.

When the routine is part of a file located in the file space or in a cartridge, the first six characters in the alpha register must contain the names of the directory and the file in the form *DIRFIL* where *DIR* is the directory name and *FIL* is the file name (for example, *MEM + SB*).

When the routine is in program memory, the first three characters in the alpha register must contain the characters **PGM** before **RUN** is executed. Any remaining characters in the register are ignored.

**Note:** If the routine expects a number to be in the numeric display register, you must enter the number before executing this instruction.

## Chapter 9: Cassette Tape Operations

---

This chapter describes how to use the optional CI-7 cassette interface cable, which lets you connect the TI-95 to a cassette tape recorder. You can then use the recorder to store and retrieve files on tape. You can perform tape operations directly from the keyboard or from within a program.

---

<b>Table of Contents</b>	Introduction .....	9-2
	Selecting a Cassette Recorder and Tapes .....	9-3
	Guidelines for Good Recording .....	9-4
	Connecting Your Recorder to the TI-95 .....	9-6
	Finding the Correct Volume Setting .....	9-7
	Using the <b>TAPE STORAGE</b> Menu .....	9-8
	Writing a File on Tape .....	9-9
	Reading or Verifying a File on Tape .....	9-12
	Examples of Tape Operations .....	9-16
	Performing Tape Operations in a Program .....	9-18
	Reference Section .....	9-20

© 2010 Joerg Woerner  
Dataman, Clickable Museum

Cassette tape operations are similar to the file operations described in the previous chapter. You can store either program files or data files on a tape and then retrieve them whenever they are needed.

---

### Types of Files

You can create two types of files.

- ▶ A program file is a copy you make of the contents of program memory.
- ▶ A data file is a copy you make of the contents of a series of data registers.

After a file is stored on tape, you must first load it back into the calculator's memory before you can use it.

**Note:** There is one difference between a program file stored on tape and one stored in the calculator's file space. In the file space, a program file does not have to be reloaded before you can use it.

### File Names

When you create a file, you must specify a three-character name for the file. The name can include letters, numerals, and punctuation symbols.

You must identify a data file by using a + as the first character of the file name (for example, +SQ). You cannot use + as the first character for a program file. This lets the calculator distinguish between a program file and a data file.

When entering a file name, you can use the **CLEAR** key to erase the current name and start over.

### Types of Tape Operations

You can perform three types of tape operations. These are:

- ▶ Writing (storing) a program or data as a file on tape.
- ▶ Reading (loading) a program or data back into memory.
- ▶ Verifying that the program or data in memory is identical to the tape file. (Normally, you should verify a file after writing it to tape. This ensures that the file was written properly.)

## Selecting a Cassette Recorder and Tapes

---

The CI-7 cassette interface cable and the TI-95 are compatible with most standard cassette recorders and tapes. However, some types of recorders and tapes give better performance than others. Use the following information to help you select the types that give you the best results.

---

### Selecting a Recorder

In order to properly connect the CI-7 cassette interface cable, you should select a recorder that has a:

- ▶ Microphone jack.
- ▶ Earphone or external speaker jack.
- ▶ Remote control jack.

For better performance and extra convenience, you may want to select a recorder that has a:

- ▶ Digital tape counter. (When you have more than one file on the same tape, the counter lets you keep track of the location of each file.)
- ▶ Optional AC adapter. (This avoids problems that are often caused by weak batteries, such as uneven recording or playback speeds.)

### Selecting Tapes

To ensure maximum reliability, use the following guidelines when you select cassette tapes.

- ▶ Choose tapes with low-noise characteristics. Tapes with extended frequency response, such as digital tapes, are unnecessary and cost more than common audio cassettes.
- ▶ Choose high-quality cassette tapes. Low-quality tapes tend to break or tangle.
- ▶ Use the type and length of tape recommended by the manufacturer of your recorder.

## Guidelines for Good Recording

---

The following guidelines can help you avoid potential problems that may occur when you perform tape operations.

---

### Tips for Recording on Tape

When you record a file on a cassette tape:

- ▶ Do not record on the leader (the non-magnetic segment at the beginning of the tape).
- ▶ Do not record too close to the end of the tape. If the tape runs out before the recording is complete, the entire file is unusable.
- ▶ Leave a few seconds of space between files that you record on consecutive sections of the tape. If a new file overlaps an old one, the old file is no longer usable.
- ▶ Record and play tape files on the same cassette recorder. Because of calibration differences between recorders, files recorded on one recorder may not read reliably on another one.

### Using the Tape Counter

The reading on a tape counter is useful only if you ensure that zero corresponds to the beginning of the tape. If your cassette recorder has a tape counter, always be sure the counter is set properly before you use it.

- ▶ Rewind the tape to the beginning.
- ▶ Set the tape counter to zero.

Each time you write a file on tape, be sure to note the counter setting at the beginning and at the end of the file. By noting both settings, you can perform tape operations more quickly and efficiently.

- ▶ To read or verify an existing file, you need to find the beginning of the file.
- ▶ To write a new file following an old one, you need to know where the old file ends so that you do not overlap the two files.

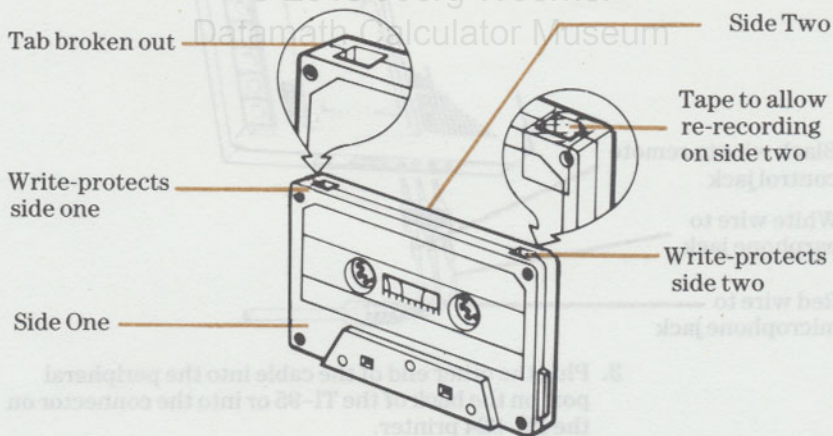
### Making a Copy of a File on Tape

If you want to copy a file from one cassette tape to another, you should always read the file into the TI-95 and then write the file on the new tape.

Duplicating a tape file by connecting one recorder to another may affect the accuracy of the copy.

### Preventing Accidental Erasure

If a tape contains files that you want to keep permanently, you may want to "write-protect" the tape to prevent you from accidentally erasing any of the files. To write-protect each side of the tape, break off the left tab on the back of the tape. When the tab is removed, you cannot press the RECORD button on the recorder.



**Note:** If you need to record on a tape that has a tab missing, place a piece of cellophane tape over the tab opening.

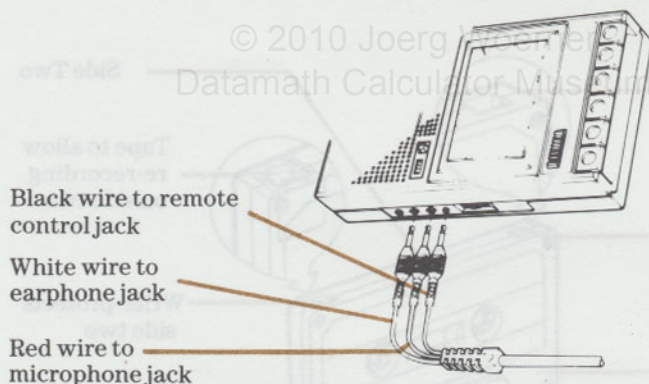
## Connecting Your Recorder to the TI-95

The CI-7 cassette interface cable contains electronic circuitry that enables the TI-95 to exchange information with your recorder. Always make sure the calculator and recorder are properly connected before you begin any tape operations.

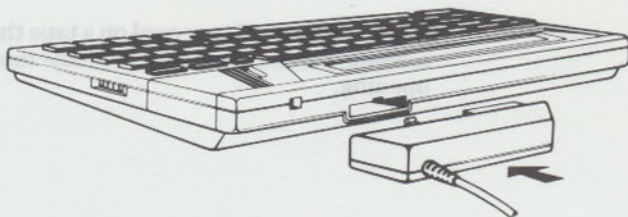
### Connecting the Recorder

To connect your cassette recorder to the TI-95:

1. Be sure the TI-95 is turned off.
2. Insert the three plugs at one end of the CI-7 cassette interface cable into your recorder as described below.
  - ▶ Insert the plug on the red wire into the microphone jack (usually labeled MIC).
  - ▶ Insert the plug on the white wire into the jack for the earphone, monitor, or external speaker (usually labeled EAR or MONITOR).
  - ▶ Insert the plug on the black wire into the remote control jack (usually labeled REM).



3. Plug the other end of the cable into the peripheral port on the back of the TI-95 or into the connector on the PC-324 printer.



The volume and tone settings that you normally use when listening to cassette tapes may not work for tape storage. Although the tone should work fine on a medium setting, different recorders may require different volume settings. Before you attempt to store or retrieve important files, find the setting that works best for your recorder.

---

### Method

When you first use the procedures in this chapter, use the following instructions to experiment with different volume settings. For most recorders, one volume setting works fine for all TI-95 cassette operations. (With some types of recorders, the recording level is set automatically and is not affected by the setting of the volume control.)

To find the correct volume setting for your recorder, use a small program or a series of data registers as a test file.

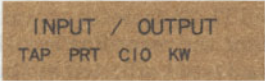
1. Adjust the volume control to its highest setting.
2. Write the test file on tape.
3. Using the same volume setting, read or verify the file.

- If the message **FILE aaa READ** or **FILE aaa OK** is displayed (where *aaa* is the name of the file), note the volume setting and use it for all subsequent tape operations.
- If the message **CASSETTE ERROR** is displayed, lower the volume slightly and repeat the procedure beginning at step 2. (If your recorder has automatic recording level, you only need to lower the volume and read or verify the file again.)

**Note:** If the above procedure does not work, you may need to use a different volume setting for writing files than you use for reading them. Experiment until you find the settings that work with your recorder.

The TAPE STORAGE menu lets you choose the type of tape operation you want to perform. However, this menu cannot be displayed directly from the keyboard. You must first display the INPUT/OUTPUT menu and then select the option for tape operations.

**Selecting Tape Operations** Before you can perform a tape operation, you must first press **I/O** to display the **INPUT/OUTPUT** menu.



INPUT / OUTPUT  
TAP PRT CIO KW

From this menu, select **<TAP>** to display the **TAPE STORAGE** menu shown below.

**The TAPE STORAGE Menu**

When you select **<TAP>** from the **INPUT/OUTPUT** menu, the calculator displays:



TAPE STORAGE  
RD WRT VFY

**<RD>** Reads a program or data file that was previously stored on tape and loads it into the calculator's memory.

**<WRT>** Writes (saves) a program or data file onto tape.

**<VFY>** Compares the program or data in memory with a file stored on tape, and then verifies if the two are identical.

Each of these operations is discussed on the following pages.

The procedure on this and the following pages describes how to write (store) a file on tape. You can write either a program file or a data file.

### Initial Steps

To begin writing a file on tape:

1. Press **I/O** <TAP> to display the **TAPE STORAGE** menu.
2. Select <WRT> from the menu.

The calculator displays:



SAVE PGM OR REGS  
PGM REG ESC

3. Press the applicable key for the type of file you want to write.
  - For a program file, press <PGM> and proceed to "Entering the File Name" on the next page.
  - For a data file, press <REG> and proceed to the top of next page.

### If the File Is a Data File

If you selected <REG>, the calculator displays:

ENTER # OF REGS  
ENT ESC

1. Specify both the number of registers to be written and the address of the first register by entering a number. The number must be in the form:

*nnn.sss*

where *nnn* specifies the number of registers, and *sss* specifies the starting register. You must include leading zeros in *sss*. If you do not specify *sss*, the calculator assumes the starting register to be 000.

2. Press <ENT> to enter the number.

The calculator then prompts you to enter the name of the file.

### Entering the File Name

After you specify the type of file (and, if necessary, the data registers to be written), the calculator displays:

ENTER FILE= \_\_\_\_\_  
ENT ESC

1. Enter a three-character name for the file. (If you specified a data file, a + is automatically displayed as the first character in the name.)
2. Press <ENT>.

The calculator then prompts you to begin operating the recorder.

## Operating the Recorder

Before you begin using the recorder, set the volume, tone, and tape counter to their correct positions.

The following table lists the sequence of prompts that are displayed when you are writing the file on tape. The I/O indicator appears when **POSITION TAPE** is displayed and remains visible until **PRESS STOP** appears.

POSITION TAPE OK ESC	Use the fast forward and rewind controls to position the tape where you want to write the file. Then press <OK>.
PRESS RECORD OK ESC	Press the RECORD button. (On some recorders, you must press the PLAY and RECORD buttons at the same time.) Then press <OK>.
WRITING . . .	The file is being recorded on the tape.
PRESS STOP OK	The recording is completed. Press the STOP button. Then press <OK>.
FILE aaa WRITTEN RD WRT VFY	Confirms that file <i>aaa</i> has been stored. Be sure to record the setting of the tape counter.

Anytime a tape operation menu contains the <ESC> selection, you can press <ESC> to cancel the operation.

## Reading or Verifying a File on Tape

---

The procedure on this and the following pages describes how to read (load) or verify a file on tape. You can use either a program file or a data file.

---

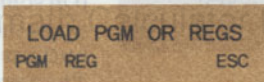
### Initial Steps

To begin reading or verifying a file on tape:

1. Press **[F10]** <TAP> to display the **TAPE STORAGE** menu.
2. Select the applicable key for the type of tape operation you want to perform.
  - To read a file, press <RD>.

- To verify that a tape file matches the contents of memory, press <VFY>.

The calculator displays a menu that lets you select the type of file you want to use. If you pressed <RD>, for example, the calculator displays:



LOAD PGM OR REGS  
PGM REG ESC

3. Press the applicable key for the type of file you want to use.
  - For a program file, press <PGM> and proceed to "Entering the File Name" on the next page.
  - For a data file, press <REG> and proceed to the top of the next page.

**If the File Is a Data File** If you selected <REG>, the calculator displays the screen shown below.

ENTER 1st REG #  
ENT ESC

Enter the number of the starting register into which the data is to be loaded or from which it is to be verified, and press <ENT>.

**Entering the File Name** After you specify the type of file (and, if necessary, the starting data register), the calculator displays:

ENTER FILE= \_\_\_\_\_  
ENT ESC

1. Enter a three-character name for the file. (If you specified a data file, a + is automatically displayed as the first character in the name.)
2. Press <ENT>.

The calculator then prompts you to begin operating the recorder.

(continued)

## Reading or Verifying a File on Tape (Continued)

### Operating the Recorder

Before you begin using the recorder, set the volume, tone, and tape counter to their correct positions.

The following table lists the sequence of prompts that are displayed when you are reading or verifying the file on tape. The **I/O** indicator appears when the **POSITION TAPE** message is displayed and remains visible until **PRESS STOP** appears.

POSITION TAPE OK ESC	Use the fast forward and rewind controls to position the tape to the beginning of the file. Then press <OK>.
PRESS PLAY OK ESC	Press the PLAY button. Then press <OK>.
SEARCHING . . .	The calculator is searching for the beginning of the file.
READING . . .	The file has been found and is being read or verified.
VERIFYING . . .	
PRESS STOP OK	The operation is completed. Press the STOP button and then press <OK>.
FILE aaa READ RD WRT VFY	Confirms that file <i>aaa</i> has been read or verified.
FILE aaa OK RD WRT VFY	

If you do not see either message indicating the operation is complete, refer to the next page.

## Operating the Recorder (Continued)

If the first file encountered after you pressed PLAY has a different name than the one you specified, the calculator displays:

FOUND=aaa, CONT?  
YES NO

(where *aaa* is the name of the file)

- ▶ To continue searching for the specified file, press <YES>.
- ▶ To cancel the search, press <NO>. The calculator then prompts you to stop the recorder and press <OK>.

After you press <OK>, the message **WRONG FILE FOUND** is displayed. You must press **CLEAR** to continue.

## If You Have Problems

Several factors can affect the success of a read or verify operation.

- ▶ The cassette interface cable may not be correctly connected to the calculator or to the recorder.
- ▶ The volume and tone settings on the recorder may need to be adjusted.
- ▶ The tape may have been positioned incorrectly.

If an error message appears, press **CLEAR** and then retry the procedure. (For a list of error messages, refer to Appendix B in the *TI-95 User's Guide*.)

If the calculator is unable to identify the file you specify, it continues to search. You cannot interrupt this process except by pressing the **RESET** button. If the search is unsuccessful, the calculator terminates the search after a period of time.

## Examples of Tape Operations

These examples demonstrate how to write and read a data file on a cassette tape.

### Example: Writing a Data File

Use the following sequence to store three values in data registers 010, 011, and 012 and write them on tape as a data file.

Before you begin using the recorder, be sure to set the volume, tone, and tape counter to their correct positions.

Procedure	Press	Display
Clear data registers	<b>2nd</b> <b>[CMS]</b>	0.
Store three values	10 <b>[STO]</b> 010 11 <b>[STO]</b> 011 12 <b>[STO]</b> 012	10. 11. 12.
Display <b>TAPE STORAGE</b> menu	<b>[I/O]</b> <b>&lt;TAP&gt;</b>	TAPE STORAGE
Write from registers	<b>&lt;WRT&gt;</b> <b>&lt;REG&gt;</b>	ENTER # OF REGS
3 registers, starting with register 010	3.010 <b>&lt;ENT&gt;</b>	ENTER FILE = + __
Name the file <b>+ SP</b>	<b>SP</b> <b>&lt;ENT&gt;</b>	POSITION TAPE
Position tape; write down counter setting	<b>&lt;OK&gt;</b>	PRESS RECORD
Press <b>RECORD</b> button	<b>&lt;OK&gt;</b>	WRITING...
		PRESS STOP
Press <b>STOP</b> button; write down counter setting	<b>&lt;OK&gt;</b>	FILE + SP WRITTEN

**Example:**  
**Reading a**  
**Data File**

Use the following sequence of keystrokes to read the three data values you saved into registers 020, 021, and 022.

Before you begin using the recorder, be sure to set the volume, tone, and tape counter to their correct positions.

Procedure	Press	Display
Clear display and data registers	<b>CLEAR</b> <b>2nd</b> <b>[CMS]</b>	0.
Display <b>TAPE STORAGE</b> menu	<b>I/O</b> <b>&lt;TAP&gt;</b>	TAPE STORAGE
Read to registers	<b>&lt;RD&gt;</b> <b>&lt;REG&gt;</b>	ENTER 1st REG #
Specify register 20	<b>20 &lt;ENT&gt;</b>	ENTER FILE = + ____
Specify data file <b>+SP</b>	<b>SP &lt;ENT&gt;</b>	POSITION TAPE
Position tape to start of file	<b>&lt;OK&gt;</b>	PRESS PLAY
Press <b>PLAY</b> button	<b>&lt;OK&gt;</b>	SEARCHING... READING... PRESS STOP
Press <b>STOP</b> button	<b>&lt;OK&gt;</b>	FILE + SP READ
Confirm load was OK	<b>RCL</b> 020	10.
	<b>RCL</b> 021	11.
	<b>RCL</b> 022	12.

## Performing Tape Operations in a Program

You can design a program so that it performs its own tape operations. After you are familiar with using the tape operations from the keyboard, read this section to learn how to use them within a program.

### Important Concept

When you perform tape operations within a program, you use a different sequence of keystrokes.

In the learn mode, the calculator prompts you only for the type of tape operation; it does not display menus for other information, such as the file name. (When you run the program, however, the calculator still displays prompts for operating the recorder.)

Within the program, you must specify:

- ▶ Which data registers are to be used (if the file is a data file).
- ▶ The name and type of the file.

### Specifying the Data Registers

For a program that writes data, you must ensure that the value *nnn.sss* (as described on page 9–10 of this guide) is in the numeric display register before the program executes <WRT>.

For a program that reads or verifies data, you must ensure that the number of the starting register is in the numeric display register before the program executes <RD> or <VFY>.

Use this section as a source of reference information for tape operations.

**Specifying the Name and Type of File** To specify a file name, enter the file name immediately after the tape operation key. If you use fewer than three characters, the calculator adds trailing spaces. (You do not activate the alpha mode to fill in this field.)

If the file is a data file, you must include the required + as the first character of the file name.

**Sample Instruction Sequences** The following examples show instruction sequences typical of those you might use for performing tape operations within a program.

**[I/O] <TAP>** Writes 10 registers (starting with register 002) as a data file named  
**10.002 <WRT> + GP** + GP.

**[I/O] <TAP>** Reads the data file named + CA  
**5 <RD> + CA** into data registers, starting at register 005.

**[I/O] <TAP>** Verifies that the program in  
**<VFY> MTK** memory matches the program file named MTK.

When the program performs any of these key sequences, the calculator then prompts you to begin operating the recorder.

Use this section as a source of reference information for tape operations.

---

**Tape Functions** **[I/O] <TAP>**—Displays the **TAPE STORAGE** menu. The menu lets you write (<WRT>) programs and data as named tape files, as well as read (<RD>) or verify (<VFY>) files you have written. All the functions, whether used as keyboard commands or program instructions, show prompts (such as **PRESS RECORD**) for operating the recorder.

The write function saves a series of data registers or the program contained in memory as a tape file.

The verify function compares a tape file with a series of data registers or the program contained in memory. After the comparison, the calculator indicates whether the file matches the contents of memory.

The read function loads a tape file into memory, replacing the data or program previously in memory. If there is insufficient room for the file, the read is not performed and the contents of memory are unchanged.

Before carrying out a read or verify function, the calculator must find the named file on the tape. While searching for the file, the calculator displays the message **SEARCHING**. If the calculator is unable to find the file immediately, it will continue to search for at least 30 seconds before displaying an error message. During the search, you can reposition the tape if necessary. If you realize that you have made a mistake and you prefer not to wait for an error message to appear, pressing the **RESET** button cancels the function. If a file with a different name is found, the calculator lets you decide whether to continue searching for the original file.

**Note:** The calculator cannot detect whether a recorder is connected. If the recorder is not connected, any attempt to read or verify eventually displays an error message because the search is unsuccessful. A write operation, however, may appear to be successful.

**Program Tape Files** A program tape file must be assigned a three-character file name. The first character of the name cannot be a + because that character identifies a data file.

When you write a program file, the entire program is written, starting with the instruction at program step 0000 and ending with the last instruction in the program. Because a program is written in multiples of eight bytes, the calculator might include a few NOP instructions at the end of the file. When you read a program file into program memory, program memory is cleared and the file is loaded into memory starting at program address 0000.

**Data Tape Files** A data tape file must be assigned a three-character file name. The first character of the name must be a +. The + gives the calculator a way to identify the file as a data file.

**Keyboard Commands** If you use the **TAPE STORAGE** menu from the keyboard, the calculator prompts for information such as a file name or the number of registers.

If you specify fewer than three characters for a file name, the calculator supplies trailing spaces. If you make a mistake while entering a file name, you must press **CLEAR** to restart the entry.

**Program Instructions** If you use the **TAPE STORAGE** menu while the calculator is in the learn mode, the calculator does not prompt for file names or required numeric information. Instead, you must supply the required information as either a field following the function or a number in the numeric display register.

When you execute the tape function as a program instruction, the function uses the information you stored.

(continued)

**Write Program** **File to Tape:** **Keyboard** **Command** **IO** <TAP> <WRT> <PGM> *aaa* <ENT>—Begins the sequence for writing the program contained in program memory as a tape file named *aaa*. The first character of file name *aaa* must not be a + because that character identifies a data file.

**Write Data** **File to Tape:** **Keyboard** **Command** **IO** <TAP> <WRT> <REG> *nnn.sss* <ENT> + *aa* <ENT>—Begins the sequence for writing a series of data registers as a tape file named +*aa*. The calculator supplies the required + as the first character of file name +*aa*. The number *nnn.sss* specifies the number of registers to be written (*nnn*) and the address of the starting register (*sss*). The number *nnn* must be greater than zero. You must include any leading zeros in *sss*. If you omit *sss*, the calculator uses 000 as the starting register.

**Read Program** **File from Tape:** **Keyboard** **Command** **IO** <TAP> <RD> <PGM> *aaa* <ENT>—Begins the sequence for reading the program file named *aaa* from the tape into program memory. The first character in the file name must not be a +.

**Read Data** **File from Tape:** **Keyboard** **Command** **IO** <TAP> <RD> <REG> *sss* <ENT> + *aa* <ENT>—Begins the sequence for reading the data file named +*aa* from the tape into a series of data registers. The calculator supplies the required + as the first character of +*aa*. The file contents replace any data already in the destination registers, starting at register *sss*. Register *sss* need not be the same register from which the data was written.

(continued)





**Verify** **Program File:** **Keyboard Command** **I/O** <TAP> <VFY> <PGM> *aaa* <ENT>—Begins the sequence for comparing the program tape file named *aaa* with the contents of program memory, starting at step 0000. The first character in the file name must not be a +. The file is used only for comparison. After the comparison, the calculator displays a message indicating whether the file matches the program memory. When the file matches but memory contains additional instructions beyond the area compared, the message **MORE DATA IN PGM** is displayed.

**Verify Data** **File:** **Keyboard Command** **I/O** <TAP> <VFY> <REG> *sss* <ENT> + *aa* <ENT>—Begins the sequence for comparing the data tape file named + *aa* with a series of data registers. The calculator supplies the required + as the first character of + *aa*. The file is used only for comparison. The comparison is made starting at register *sss*. The calculator displays a message indicating whether the file matches the contents of the data registers.

For a verify that follows writing the file, register *sss* should be the starting register from which the data was written.

For a verify that follows reading the file, register *sss* should be the starting register into which the data was read.

(continued)

- Write Program**  <TAP> <WRT> *aaa*—Begins the prompting sequence for writing the program in program memory to the tape as a file named *aaa*. The first character of *aaa* must not be a +.
- File to Tape:** *aaa*
- Program** *aaa*
- Instruction** *aaa*
- Write Data**  <TAP> <WRT> + *aa*—Begins the prompting sequence for writing a series of data registers as a tape file named +*aa*. The first character of the file name must be a +.
- File to Tape:** +*aa*
- Program** +*aa*
- Instruction** +*aa* When the instruction is executed, the numeric display register must contain a number in the form *nnn.sss*, specifying the number of registers to be written (*nnn*) and the address of the starting register (*sss*). The number *nnn* must be greater than zero. You must include any leading zeros in *sss*. If you omit *sss*, the calculator uses 000 as the starting register.
- Read Program**  <TAP> <RD> *aaa*—Begins the prompting sequence for reading program file *aaa* from the tape into program memory. The first character in the name *aaa* must not be a +.
- File from Tape:** *aaa*
- Program** *aaa*
- Instruction** *aaa*
- Read Data**  <TAP> <RD> + *aa*—Begins the prompting sequence for reading the data file named +*aa* into a series of data registers. You must use + as the first character of the file name. When the instruction is executed, the numeric display register must contain a number in the form *sss* specifying the starting register into which the file data is to be read. Register *sss* need not be the same register from which the data was written. The file data replaces any data already in the registers.
- File from Tape:** +*aa*
- Program** +*aa*
- Instruction** +*aa*

---

**Verify  
Program File:  
Program  
Instruction**

**[V]** <TAP> <VFY> *aaa*—Begins the prompting sequence for comparing the program tape file named *aaa* with the contents of program memory. The first character in the file name must not be a +. The file is used only for comparison.

If the file matches and all program memory beyond the program contains NOP instructions, the calculator displays the message **FILE *aaa* OK**.

If the file does not match or if a problem occurs, the calculator displays a corresponding error message and halts the program.

**Note:** The message **MORE DATA IN PGM** indicates that the file matches but memory contains additional instructions beyond the area compared.

**Verify  
Data File:  
Program  
Instruction**

**[V]** <TAP> <VFY> +*aa*—Begins the prompting sequence for comparing the data file named +*aa* with a series of data registers. You must use + as the first character of the file name. When the instruction is executed, the numeric display register must contain a number in the form *sss* specifying the starting register where the comparison is to begin. Register *sss* should be the same register from which the data was written or into which it was read. The file is used only for comparison.

If the file matches the contents of the specified registers, the calculator displays the message **FILE *aaa* OK**.

If the file does not match or if a problem occurs, the calculator displays a corresponding error message and halts the program.



# Chapter 10: Developing Your Own Programs

---

This chapter describes how to use the programming concepts discussed in previous chapters to develop your own programs.

---

<b>Table of Contents</b>	<b>Programming Considerations .....</b>	<b>10-2</b>
	<b>Planning Your Program .....</b>	<b>10-6</b>

The next few pages contain reminders and some general information you should keep in mind while developing your own programs.

## Setting Initial Program Conditions

Programs usually include an initialization routine to establish a known set of conditions. Such a routine may, for example, set calculator parameters and store initial values for the program.

Typically, the initialization routine resides at the beginning of the program and is executed only once.

Although the size and complexity of the routine depend on the complexity of the program, you should consider at least:

- ▶ Setting default conditions (**HELP**). (Executing **HELP** in a program is identical to pressing **HELP** <YES> from the keyboard.)
- ▶ Setting partitions for user memory (**2nd** **[PART]**). This may not be necessary if you use **HELP**.
- ▶ Clearing any pending operations, clearing the display, and cancelling scientific notation (**CLEAR**). This is not necessary if you use **HELP**.
- ▶ Clearing (resetting) user flags 00–14 (**FLAGS** <CLR>).
- ▶ Clearing data registers (**2nd** **[CMS]**) or storing initial values in specific registers.

## Example

As an example of an initialization routine, the following program segment clears all data registers, selects the decimal number base, clears flags 00–14, and stores the value 12 in data register A.

PC =	Program Mnemonics	Comments
0000	CMS	Clear data registers
0001	DEC	Select decimal format
0002	CFG	Clear flags 00–14
0003	12 STO A	Initialize register A

**AOSTM Considerations** As explained in the *TI-95 User's Guide*, elements of an expression are not necessarily evaluated in the same sequence in which you enter them.

For example, the AOS feature evaluates the sequence  $RCL\ 001 + 3 * 6$  as  $RCL\ 001 + (3 * 6)$ . If you want the expression to be evaluated differently, such as  $(RCL\ 001 + 3) * 6$ , include the parentheses in the program.

An immediate function operates only on the displayed value; it does not complete a pending operation before executing.

For example, if you attempt to evaluate the expression  $(2 * 3)^2$  by entering  $2 \times 3 \boxed{x^2}$ , the expression is instead evaluated as  $2 * 3^2$ . To avoid this problem, enter the expression as either  $(2 \times 3) \boxed{x^2}$  or  $2 \times 3 \boxed{= \boxed{x^2}}$ .

Keep in mind that if you use  $\boxed{=}$  in a subroutine, it completes all pending operations, including any that were pending when the subroutine was called.

**Angle-Unit Considerations** Before using a trigonometric function in a program, you must ensure that the appropriate angle unit has been selected. You can use the following table for setting angle units to a known condition from within a program.

Key Sequence	Sets Angle Mode To
$\boxed{INV} \boxed{2nd} \boxed{[DRG]}$	Degrees
$\boxed{INV} \boxed{2nd} \boxed{[DRG]}$ $\boxed{2nd} \boxed{[DRG]}$	Radians
$\boxed{INV} \boxed{2nd} \boxed{[DRG]}$ $\boxed{2nd} \boxed{[DRG]}$ $\boxed{2nd} \boxed{[DRG]}$	Grads

(continued)

**Using Labels Correctly** Using program labels simplifies writing and modifying a program. When assigning labels, however, make sure to use a different label for each section of the program.

If you inadvertently assign the same label to two segments of a program, transfer instructions that refer to that label will always transfer control to the first occurrence of the label. You cannot transfer control to another segment that has the same label.

**Avoiding Memory Conflicts** If your program stores an alpha message in memory, keep in mind that each message stored—even a short message—occupies 10 contiguous data registers. If you do not allow a sufficient number of registers for a message, it could conflict with other data.

For example, if you have stored a numeric value in data register 018 and then store an alpha message starting at register 015, the numeric value will be overwritten by a portion of the message.

You should also keep in mind that the <QAD> and <CUB> selections of the **EXTENDED FUNC** menu use data registers. For information on the specific registers involved, refer to the “Math Operations” chapter in the *TI-95 User’s Guide*.

By planning specific locations for each numeric value and each alpha message, you can avoid such conflicts.

You do not need to avoid conflicts between data registers and the registers used for built-in statistical functions; statistical registers reside in a separate area of the TI-95 memory.

(continued)

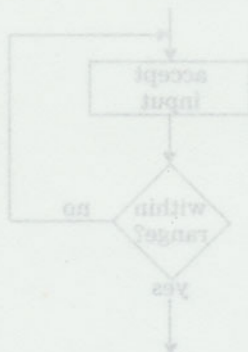
Although it is possible to write a program using the "trial-and-error" method, planning gives you several advantages. It minimizes rewriting and generally results in a more efficient program. A well-planned program is also easier to modify and is less likely to contain errors.

### Saving the t-Register Contents

A few of the calculator's functions, such as polar/rectangular conversions, leave a result in the t-register or expect an input value to be stored there. Each time your program executes such a function, the previous contents of the t-register are lost. If this causes a problem, you can save the t-register contents in a data register.

As an example, the following program segment saves the contents of the t-register in data register 005 without altering either the t-register or the display.

PC =	Program Mnemonics
0000	$\sim t$
0001	STO 005
0004	$\sim t$



# Planning Your Program

---

Although it is possible to write a program using the “trial-and-error” method, planning gives you several advantages. It minimizes rewriting and generally results in a more efficient program. A well-planned program is also easier to modify and is less likely to contain errors.

---

**Step 1:** Clearly defining the problem minimizes the time you will spend developing a solution. In defining the problem, try asking the following questions.

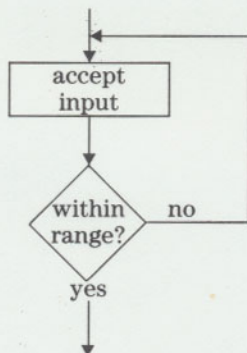
**Define the Problem**

- ▶ What elements of the problem are already known?
- ▶ What information must be supplied from the keyboard as the program runs?
- ▶ What elements of the problem must the program calculate?
- ▶ What types of calculations will be necessary?

**Step 2:**  
**Develop**  
**a Solution**

A complex problem can be simplified by breaking it down into “modules,” or program segments, that each deal with a specific task in the overall solution.

A device called a flowchart can help you design modules and show relationships between modules. A flowchart uses symbols to illustrate the planned solution. Generally, a rectangle is used to indicate a task and a diamond is used to indicate a test.



**Step 3:**

**Determine Messages and Prompts**

Try to design messages so that they consist of 16 or fewer characters. Although you can create a message of as many as 80 characters, short messages are easier to read.

You may find it useful to draw templates such as the one below to record planned messages, columns for merged numbers, and function-key labels.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16				
F1				F2				F3				F4				F5			

**Step 4:**

**Determine Memory Allocation**

Large programs (and some small programs that generate long lists of data) require careful allocation of available memory.

At this stage of planning, you should:

- Determine the approximate amount of memory your program will occupy. Plan to partition enough program memory to allow for possible modification of the program.
- Determine the number of registers the program will need for stored alpha messages and for numeric values, such as counters and results of calculations. Record the planned location of each message and value. Plan to partition enough data registers.

(continued)

## Planning Your Program (Continued)

After developing a solution and deciding on messages and memory locations for program data, you are ready to translate your solution into a sequence of program instructions.

### Step 5: Write the Program Steps

You should now be ready to write the program on paper, translating each symbol from the flowchart into a sequence of instructions. Writing the planned program steps on paper lets you use your own form of “shorthand” and lets you include comments about what the program does at each point.

⋮		
LABEL AA		POINTER-ADJUST ROUTINE
recall 015		Get pointer
+ recall 016 =		Add offset
store 017		Store pointer + offset
⋮		

You can use these written steps to mentally test the solution before entering the keystrokes. If you find that you are repeating the same sequence of program instructions, consider using the sequence as a subroutine.

### Step 6: Enter the Keystrokes

When you are satisfied that the program instructions will accomplish the program's task, the next step is to enter the keystrokes.

While translating the written steps into keystrokes, remember to use the **2nd** and **INV** keys as necessary. For example, if you have written an instruction as LABEL AA, you must press **2nd** **[LBL]** AA. For a listing of instruction mnemonics and their corresponding keystrokes, refer to Appendix C.

(continued)

---

**Step 7: Test and Edit the Program** After you have entered the keystrokes for your program, test the program by running it. In many cases, the program will contain errors and will not run correctly the first time.

If the program does not stop after a reasonable time, press **BREAK**, and then press **LEARN** <PC> to display the part of the program that was executing when you pressed **BREAK**. (You can also insert BRK as a program instruction to periodically check the progress of the program.) After exiting the learn mode, you can turn on the trace function and press <GO> to resume program execution. When trace is on, the calculator displays the program mnemonics and the audit trail as the program runs. If you have a printer connected, the mnemonics and audit trail are also printed.

You can activate trace either from the keyboard or from within the program itself. If you have isolated the problem to a particular segment of the program, insert a TRC instruction before the segment and an INV TRC instruction after the segment.

If you only need to see a specific value or message to determine that your program is running correctly, you may prefer to insert a PRT or a PAU instruction rather than tracing each instruction.

If you want to examine a labeled instruction or an instruction whose program step you know, you can use **2nd** **[GTL]** or **INV** **2nd** **[GTL]** from the keyboard to set the program counter to the desired step. Then press **LEARN** <PC> to display the instruction.

Continue to test and edit until the program runs correctly. You should test the program using "worst case" data to ensure that it can handle a wide range of situations.

(continued)

### Step 8: Save and Document the Program

When your program is finished, you may want to save it as a file (either in the calculator's file space or in a storage device such as a Constant Memory™ cartridge).

You should also prepare any written documentation for using and maintaining the program. Your written notes and program steps can be useful. If you have a printer, print a listing of the program and its labels.

If you will be using the program infrequently or if others will use the program, it is a good idea to compile written instructions for its use. This is particularly important with a complex program that requires a great deal of input from the keyboard.

© 2010 Joerg Woerner

Datamath Calculator Museum

(continued)

# Appendix A: Advanced Memory Functions

---

This appendix is for experienced programmers who want direct access to the system memory of the TI-95. To use this information, you should be experienced in using hexadecimal numbers and conducting operations that involve the manipulation of bytes.

---

<b>Table of Contents</b>	<b>Introduction</b> .....	<b>A-2</b>
	<b>Changing the System-protection Mode</b> .....	<b>A-4</b>
	<b>Storing or Recalling a Single Byte</b> .....	<b>A-6</b>
	<b>Using the Unformatted Mode</b> .....	<b>A-8</b>
	<b>Internal Representation of Numeric Values</b> .....	<b>A-13</b>
	<b>Accessing Assembly-language Subroutines</b> .....	<b>A-15</b>

All memory in the TI-95 is organized as a series of eight-byte registers. These include data registers and system registers. Normally, system registers are protected and you can access only the data registers. By removing system protection, however, you can gain direct access to all registers.

## The System-Protected Mode

Each time you turn on the calculator, the system is protected. In the system-protected mode, you can:

- ▶ Access the 16 user flags (00–15) using the SF, RF, and TF instructions.
- ▶ Access data registers using functions such as STO, RCL, and DSZ followed by a three-digit register address.
- ▶ Enter a program into the program memory by using the learn mode.
- ▶ Store and retrieve files in the file space or in a Constant Memory™ cartridge using the file operations.

## The System-Unprotected Mode

Besides the functions available in the system-protected mode, the system-unprotected mode lets you:

- ▶ Access the system flags (16–99) using the SF, RF, and TF instructions. (Refer to “System Flags” in Appendix C for a list of the system flags.)
- ▶ Access data registers or system registers using functions such as STO, RCL, and DSZ followed by a four-digit register address.
- ▶ Store or recall a single byte in any user memory area, system register, or Constant Memory cartridge by using the STB and RCB functions.
- ▶ Call an assembly-language subroutine by using the SBA function.

**Note:** You can use any of the system functions from the keyboard or from within a program. To store a system function as a program instruction, however, you must remove the system protection **before** entering the learn mode.

---

**Precaution  
for Using the  
Unprotected  
Mode**

The procedure described on the next page shows you how to remove the system protection.

Unless you have a specific reason for accessing a system register, always leave the calculator in the protected mode. This prevents you from inadvertently changing any system registers that may affect the operation of the calculator.

Each system register has a fixed use that is determined by the design of the calculator. If you accidentally or indiscriminately change the contents of a system register, you could alter option settings or destroy important data.

In extreme cases, you could temporarily disable the keyboard or the display. In such a case, you must press the **RESET** button to restore normal operation.

## Changing the System-protection Mode

---

This section shows you how to change the system-protection mode. You can always determine the current protection mode by looking for the SYS indicator in the display. If the indicator is displayed, the system is unprotected; if not, the system is protected.

---

### Removing System Protection

From the keyboard, you can remove system protection by using the following procedure. (You cannot remove system protection from within a program.)

1. Press the **FUNC** key.

The calculator displays:

```
EXTENDED FUNC
QAD CUB SYS
```

2. Press <SYS>.

The calculator displays:

```
UNPROTECT SYSTEM
YES NO
```

3. Press <YES> to unprotect the system.

The calculator turns on the **SYS** indicator and displays a menu that lets you choose from the available system functions.

```
SYSTEM FUNCTIONS
STB RCB SBA
```

- |       |  |
|-------|--|
| <STB> | Stores a value in a single byte.       |
| <RCB> | Recalls the contents of a single byte. |
| <SBA> | Calls an assembly language subroutine. |

Each of these system functions is discussed later in this appendix.

## Restoring System Protection

If the system is unprotected, you can restore system protection in either of two ways.

- ▶ Turn the calculator off and then back on.
- ▶ Use the **HELP** function.

When you press **HELP**, the calculator displays:

SET NORMAL MODE?  
YES NO ESC

- ▶ If you want to both restore system protection and restore other calculator settings to their default values, press **<YES>**.
- ▶ If you only want to restore system protection without affecting any other settings, press **<NO>** repeatedly until the following message appears.

CLR SYSTEM MODE?  
YES NO

Then press **<YES>**.

## Storing or Recalling a Single Byte

Two of the system functions, STB and RCB, let you store and recall single bytes in memory. The valid range for byte values is 0-255 in decimal mode (00-FF in hex mode). Regardless of the mode in use, you must express the byte address as a hexadecimal number.

### Calculating an Address in User Memory

To calculate the address of a byte in user memory, use one of the formulas given below. Then convert the address to hex. (For the address of a byte in system memory, refer to the "Memory Map" and "System RAM Usage" sections of Appendix C.)

- For a specified byte in a data register, use:

$$\text{Byte address} = 16384 - (8 * \text{Reg}) - \text{Byte}$$

where *Reg* is the number of the data register and *Byte* is in the range 1-8.

- For a specified step in program memory, use:

$$\text{Byte address} = 9184 + FS + \text{Step}$$

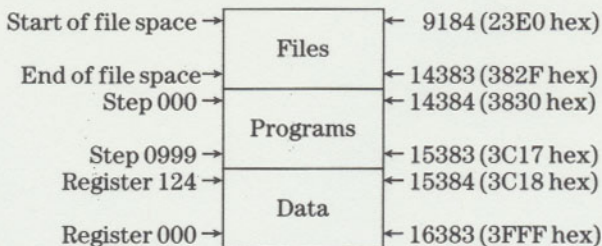
where *FS* is the number of bytes partitioned as file space and *Step* is the program step number.

- For a specified byte in the file space, use:

$$\text{Byte address} = 9183 + \text{Byte}$$

where *Byte* is in the range 1 to the total number of bytes partitioned as file space.

This illustration, based on the default partitions, shows how the formulas are derived.



## The STB and RCB Functions

<STB> *hhhh* stores the value in the numeric display in byte *hhhh*, where *hhhh* is a four-digit hex memory address.

<RCB> *hhhh* recalls into the numeric display register the value in byte *hhhh*, where *hhhh* is a four-digit hex memory address.

### Example

Use STB to store the hex value AA (170 decimal) in the third byte of data register 000 and then use RCB to recall the byte in decimal mode. First, use the equation on the previous page to calculate the address of the byte.

Byte address =  $16384 - (8 * 0) - 3 = 16381$

Converted to hex, the byte address is 3FFD.

Procedure	Press	Display
Clear and select hex mode	<b>CLEAR</b> <b>CONV</b> <BAS> <HEX>	0.
Remove protection	<b>FUNC</b> <SYS> <YES>      SYSTEM FUNCTIONS	
Enter value AA	<b>2nd</b> [ <b>A<sub>H</sub></b> ] <b>2nd</b> [ <b>A<sub>H</sub></b> ]	AA
Store at address 3FFD	<STB> 3 <b>2nd</b> [ <b>F<sub>H</sub></b> ] <b>2nd</b> [ <b>F<sub>H</sub></b> ] <b>2nd</b> [ <b>D<sub>H</sub></b> ]	AA.
Clear and select decimal mode	<b>CLEAR</b> <b>CONV</b> <BAS> <DEC>	0.
Redisplay system functions menu	<b>FUNC</b> <SYS>      SYSTEM FUNCTIONS	
Recall the value from address 3FFD	<RCB> 3 <b>2nd</b> [ <b>F<sub>H</sub></b> ] <b>2nd</b> [ <b>F<sub>H</sub></b> ] <b>2nd</b> [ <b>D<sub>H</sub></b> ]	170.
Restore protection	<b>HELP</b> <YES>      NORMAL MODE SET	

## Using the Unformatted Mode

---

The unformatted mode gives you a convenient way to treat the information in a register as a series of bytes rather than a single numeric value.

---

### What Is the Unformatted Mode?

When you enter a number in a formatted mode such as decimal or hexadecimal, the number is converted to an internal representation the calculator uses for all numeric values. (For a description of this method of representation, refer to page A-13 in this guide.) When you display a number in a formatted mode, the number is converted from the internal representation to the selected format.

In the unformatted mode, however, no conversion takes place. This gives you a way to enter a series of bytes exactly as you want them stored and lets you examine the contents of a register as a series of bytes.

You can use the same functions in the unformatted mode that you can use in the hex or octal modes. You can, for example, perform memory operations such as STO and RCL. You can also access system registers by removing system protection, just as you can in the formatted modes.

### Unformatted Entries

When you make an entry using the unformatted mode:

- ▶ Each pair of hex digits you enter represents a single byte.
- ▶ The only valid data entry keys are **0** through **9** and **2nd** [**A<sub>H</sub>**] through **2nd** [**F<sub>H</sub>**]. The calculator ignores other entry keys, such as **+/-**, **.**, and **EE**.
- ▶ If you do not enter all 16 digits, the digits you enter are left-justified and the calculator supplies trailing zeros. For example, if you enter the number 23 in unformatted mode and then press **=**, the display shows 2300000000000000.

**Unformatted Display**

When you use the unformatted mode to display a number, the calculator displays the number as a series of 16 hexadecimal digits, with each pair representing one of eight bytes.

For example, you might recall a register containing AABBCCDDEEFF0011. The first byte in the register is AA hex, the second byte is BB hex, and so on.

**Considerations for Using the Unformatted Mode**

In general, you should confine the use of the unformatted mode to entering and displaying a series of bytes. The formatted modes are better suited for entering, calculating, and displaying numeric values.

To avoid unexpected results when you use the unformatted mode, keep these considerations in mind.

- ▶ Although calculations operate normally in the unformatted mode, the displayed results are not formatted. For example, the result of  $6!$ , if displayed in the unformatted mode, is 7200000000000002.
- ▶ You could get unexpected results if you enter a series of bytes in the unformatted mode and then inadvertently display the bytes as a formatted number. For example, the unformatted entry AABBCCDDEEFF, displayed in the decimal mode, is 10, even though the entry is stored internally as AABBCCDDEEFF0000.
- ▶ Unless you are intentionally entering an unformatted number, make sure the calculator is in one of the formatted modes. This is primarily important when you are entering numbers to be used in calculations, because the exponent (described on page A-13) is not computed in the unformatted mode.

(continued)

**Example:** Use the unformatted mode to store the bytes  
**System** AB0000000000000000 in data register A. Then clear the  
**Protected** display and recall the contents of the register. To see the  
 effect of attempting to display the bytes as a decimal  
 number, select the decimal mode.

Procedure	Press	Display
Clear the display	<b>CLEAR</b>	0.
Select unformatted mode	<b>CONV</b> <b>&lt;BAS&gt;</b> <b>&lt;UNF&gt;</b>	0000000000000000
Enter the bytes	<b>2nd</b> <b>[A<sub>H</sub>]</b> <b>2nd</b> <b>[B<sub>H</sub>]</b>	AB
Store in register A	<b>STO</b> <b>A</b>	AB00000000000000
Clear the display	<b>CLEAR</b>	0000000000000000
Recall the bytes	<b>RCL</b> <b>A</b>	AB00000000000000
Attempt to display as a decimal number	<b>&lt;DEC&gt;</b>	A.B

(continued)

---

**When the  
System Is  
Unprotected**

When you remove system protection, as described on page A-4, you can access any TI-95 register—not just data registers. For this reason, when system protection is removed, functions such as STO and RCL require a four-digit field *Znnn*.

In this field, *Z* specifies both the type of register and whether the addressing is direct or indirect. *nnn* specifies the register number. Alphabetic (A–Z) addressing can be substituted for the *Znnn* field if you are addressing a data register in the range 000–025.

The following table describes how the calculator interprets a key sequence such as **RCL** *Znnn* or **STO** *Znnn*.

---

First Digit (Z)	Meaning
0	Specifies that <i>nnn</i> is a data register.
2	Specifies that <i>nnn</i> is a system register.
3	Specifies that <i>nnn</i> is a system register containing the number of another system register. (Used for indirect addressing.)
6	Specifies that <i>nnn</i> is a data register containing the number of a system register. (Used for indirect addressing.)

---

To determine system register numbers, refer to “System RAM Usage” in Appendix C.

(continued)

## Using the Unformatted Mode (Continued)

**Example:** Remove system protection and use STO with the unformatted mode to store the message **Reg 065!** (hex bytes 5265672030363521) in the first eight bytes of the alpha register (system register 065). Then use the alpha mode to display the message.

Procedure	Press	Display
Clear current alpha message	<b>ALPHA</b> <b>CLEAR</b> <b>ALPHA</b>	0.
Remove system protection	<b>FUNC</b> <SYS> <YES>	SYSTEM FUNCTIONS
Select unformatted mode	<b>CONV</b> <BAS> <UNF>	0000000000000000
Enter the byte values	5265672030363521	5265672030363521
Store in system register 065	<b>STO</b> 2065	5265672030363521
Recall as alpha message	<b>ALPHA</b> <--> <RCA> <b>=</b>	Reg 065!
Restore system protection	<b>HELP</b> <YES>	NORMAL MODE SET

(continued)

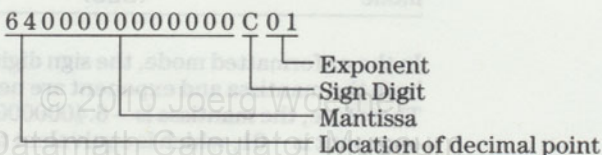
Any numeric value you enter using one of the formatted modes (including hex and octal) is converted for internal use as a floating-point decimal number. No such conversion takes place for bytes you enter using the unformatted mode.

## Floating-Point Representation

Numeric values are stored internally as three components:

- ▶ A 13-digit mantissa (with an assumed decimal point following the first digit).
- ▶ A sign digit.
- ▶ A two-digit exponent.

For example, the number  $-.64$  is stored as:



The sign digit may be one of four values, depending on the sign of the mantissa and the sign of the exponent. (Values other than those shown here are not used or indicate an error condition.)

Sign Digit	Meaning
0	Positive mantissa, positive exponent
4	Negative mantissa, positive exponent
8	Positive mantissa, negative exponent
C	Negative mantissa, negative exponent

(continued)

## Example

Use the unformatted mode to show the internal, floating-point representation of a decimal number.

Procedure	Press	Display
Enter a number	.64 <b>[+/-]</b>	-.64
Display internal representation	<b>[CONV]</b> <b>[&lt;BAS&gt;]</b> <b>[&lt;UNF&gt;]</b>	6400000000000C01
Return to decimal mode	<b>[&lt;DEC&gt;]</b>	-0.64

In the unformatted mode, the sign digit C indicates that both the mantissa and exponent are negative. Therefore, the mantissa is  $-6.400000000000$  and the exponent is  $-01$ , which is equivalent to  $-0.64$ .

Sign Digit	Meaning
0	Positive mantissa, positive exponent
4	Negative mantissa, positive exponent
8	Positive mantissa, negative exponent
C	Negative mantissa, negative exponent

(continued)

The SBA function enables you to call assembly-language subroutines. Some of the system subroutines are listed in Appendix C.

---

### The SBA Function

To access an assembly-language subroutine, press <SBA> and enter a three-digit hex value that specifies the location of the subroutine.

The first digit, with a valid range of 1 through 6, specifies whether the subroutine is located in the system ROM (read-only memory), a library cartridge, or a file directory such as the calculator's file space or a Constant Memory cartridge.

The last two digits give the pointer offset (in bytes) from the beginning of the table at the front of each chip.

Sign Digit	Meaning
1, 2, 3, or 4	Specifies a subroutine in system ROM. The system subroutines that can be useful to you are listed in Appendix C.
5	Specifies a subroutine in the currently installed library cartridge. (These subroutines are accessible only by a program in the cartridge.)
6	Specifies an assembly-language subroutine saved as a file. With this option, the last two digits are ignored. Before using the SBA function, you must place the name of the directory and file into the alpha register in the form <i>DIRFIL</i> , where <i>DIR</i> is the directory name and <i>FIL</i> is the file name (for example, MEM + SB).

(continued)

**Example** One of the system subroutines in the TI-95, number 226, displays the value contained in the numeric display register. This subroutine does not pause after showing the value as the PAU function does.

This example program counts by twos and uses routine 226 to display each count with no pause. Unprotect the system and then enter the following program.

PC =	Program Mnemonics	Comments
0000	LBL AA	
0003	+ 2 =	Counts by twos
0006	SBA 226	Displays without pause
0009	GTL AA	Loops to next count

**Running the Example** To run the example, press **RUN** <PGM>. The program starts with the number in the numeric display register and rapidly displays each count.

To stop the program, press **BREAK** or **HALT**.

# Appendix B: Advanced Input/Output Functions

---

This appendix discusses the advanced I/O (input/output) functions. The CIO (call I/O) function enables you to control the operation of a peripheral device. Before using the CIO function, you should be familiar with the information in Appendix A. By using the KW (key wait) function, your program can read individual keystrokes as they occur.

---

Table of Contents	Introduction .....	B-2
	The Parameters in a PAB .....	B-4
	Command Codes .....	B-6
	The Open Command .....	B-7
	The Read and Write Commands .....	B-10
	I/O Error Codes .....	B-11
	Performing an I/O Operation from a Program .....	B-12
	Example: Using CIO to Control a Peripheral .....	B-16
	Reading Keystrokes from a Program .....	B-22

You can use the CIO function to send an I/O command to a compatible peripheral device, such as the PC-324 Printer. (CIO cannot be used to control the CI-7 cassette interface.) This section gives an overview of how CIO uses a peripheral access block (PAB) and a data buffer to send and receive information from a peripheral device.

---

### When Do You Need the CIO Function?

You must use the CIO function to control any peripheral device that cannot be accessed directly from the calculator's keyboard.

You do not normally need to use the CIO function for a printer; the TI-95 contains built-in functions for printer operations. In some cases, however, you may want to use the CIO function to control a printer.

### What Is a PAB?

A **peripheral access block (PAB)** is an area of memory that you use to store the necessary parameters for the CIO function. These parameters include the I/O command you want to send, the device number of the device you want to access, and the address of the data buffer. You can set up the PAB at any location in memory, provided the PAB starts at the beginning of a register.

After you use the CIO function, the device returns certain parameters to the PAB. These parameters include information that tells you if the I/O operation was completed successfully.

### What Is a Data Buffer?

A **data buffer** is an area of memory that you use to store the actual data transferred between the calculator and the device. You can set up a data buffer at any location in memory.

- ▶ If your program is sending data to a device, it must store that data in the data buffer before executing the CIO function.
- ▶ If your program is receiving data, the calculator stores the incoming data in the data buffer after the CIO function is executed.

This section shows the structure of a PAB and gives a brief description of each PAB parameter. If you use the information display mode to store the parameters, you must enter each as a hexadecimal value. If you use 87B to store a parameter, you can enter the byte value using the currently selected number base. The parameters are shown here as hex values.

### Overview of Sending an I/O Command

For each I/O command that you send to a device, you use the general procedure outlined below. (Each step is described in detail later in this appendix.)

1. Build a PAB containing the parameters that apply to the I/O command you are sending.
2. If you are sending data, store that data in the data buffer.
3. Execute the CIO function to send the command.

If you want to perform the same operation repeatedly, you do not have to build a new PAB each time. However, you may need to change the data in the data buffer.

After the device responds to a command, your program should:

1. Check the error code returned by the device to see whether the operation was completed successfully or an error occurred.
2. Read any data returned to the data buffer by the device.

### Overview of Accessing a Device

Before you can send or receive information from a device, you must first send a command that "opens" the device. Once the device is open, you can perform as many I/O operations as needed. When you are finished using the device, send a command to "close" the device. Unless you close the device, any later attempt to open it will fail.

## The Parameters in a PAB

This section shows the structure of a PAB and gives a brief description of each PAB parameter. If you use the unformatted display mode to store the parameters, you must enter each as a hexadecimal value. If you use STB to store a parameter, you can enter the byte value using the currently selected number base. The parameters are shown here as hex values.

### Structure of a PAB

A PAB consists of 12 bytes that contain the various parameters. Each parameter is stored in a one-byte or two-byte field, which is filled as needed for your particular application.

**Note:** In this illustration, LSB means the **Least Significant Byte** of a byte pair.

PAB	Byte #	Parameter
	1	Device number
	2	Command code
	3	LUNO (logical unit number)
	4	Record number (LSB first)
	5	
	6	Buffer size (LSB first)
	7	
	8	Data length (LSB first)
	9	
	10	Returned status
	11	Buffer pointer (LSB first)
	12	

### Device Number

The device number specifies the device that you want to access. For example, the device number for the PC-324 printer is 12 (0C hex). To find the device number, refer to the manual that comes with the device.

### Command Code

The command code specifies the I/O command you want to send to the device. For a list of command codes, refer to page B-6 of this guide.

### Logical Unit Number (LUNO)

The LUNO field is used only for devices that may contain a number of separately addressable files on one physical unit. (This byte is ignored for other types of devices.) For example, if you want to use two or more files located on a single device, you must assign a unique, nonzero unit number to each file.

**Record  
Number**

The record number specifies a particular record in a file. This is used only with devices that allow you to access relative (random access) files. Other types of devices, such as printers, ignore these bytes.

**Buffer  
Size**

The buffer size specifies the maximum number of bytes that can be received from the device. (This does not limit the size of data you can send to the device.)

**Length  
of Data**

The length of data specifies the number of bytes of data being sent to or received from the device.

- ▶ If you are sending data to the device, you must include this value in the PAB.
- ▶ If you are not sending data, you must let the device know there is no data by specifying 0 as the data length.
- ▶ If you are receiving data from the device, the device returns the value to the PAB.

**Returned  
Status**

The returned status byte is used to indicate to you whether the I/O command was successful. A returned status byte of zero indicates no error occurred. A returned status byte that is nonzero indicates that the CIO command was not successful.

**Buffer  
Pointer**

The buffer pointer contains the address of the first (highest-numbered) byte in the data buffer.

Command codes specify the type of I/O command you want to send to a device. Not all devices, however, support all commands. If you are using a printer, for example, commands such as Read and Delete are invalid. To determine the I/O commands that apply to your device, refer to the device manual.

---

### Commonly-Used Commands

The following table lists the most commonly-used I/O commands and shows the hexadecimal code for each one. Some of these commands are discussed in more detail in this appendix.

Hex Code	Command
00	Open—Prepares a peripheral device for use.
01	Close—Terminates the use of a device.
03	Read—Reads a record from a device and stores it in the data buffer.
04	Write—Sends the data stored in the data buffer to a device.
06	Delete—Deletes a file from a mass-storage device. (You must place the name of the file in the data buffer.)
0C	Verify—Verifies that a record was read or written correctly. (This command applies only to a mass-storage device.)
0D	Format—Formats the storage medium in a mass-storage device.
FF	Reset—Closes any open files on the specified device and then resets and closes that device. (If you specify a device number of 00, the calculator closes all open files and resets all devices.)

The Open command, code 00, establishes a link between the TI-95 and the device. It also enables you to specify various file and device options.

## Information Needed for the Data Buffer

Before sending an Open command, you must place the following information into the data buffer.

- ▶ Requested I/O buffer size (2 bytes, LSB first)—Specifies the maximum length of any data you want to receive from the device. (You can specify 0000 to request the device's default value.)
- ▶ Device attributes (1 byte)—Specifies the access mode and the type of file you want to open. Refer to the next page for information concerning device attributes and how to determine the correct value for this byte.
- ▶ Device options (if any; length varies accordingly)—Specifies any device-dependent options that may be available on your device.

## Device Options

Some devices allow you to use various device options, which are usually given as a string of alphanumeric characters. To specify one of these options, use the ASCII code for each character.

On some printers, for example, the string L = D causes the printer to use double-spacing. The ASCII codes for these characters are shown here as hex numbers.

Character:	L	=	D
Hex code:	4C	3D	44

To specify this option, store 4C3D44 in the appropriate bytes of the data buffer (4C in the fourth byte, 3D in the fifth byte, and 44 in the sixth byte).

(continued)

## Device Attributes

To determine the value for the device attributes byte in the data buffer, you must first know the access mode and, if applicable, the type of file you want to open. You can then use the following table.

Bit #	Description
7,6	00—append mode (write only at end of file) 01—input mode (read only) 10—output mode (write only) 11—update mode (read or write)
5	0—sequential records 1—relative records
4	0—variable record length 1—fixed record length
3	0—display file 1—internal file
2, 1, 0	always zero

For each bit, fill in the applicable digit, 0 or 1. Then determine the value of the byte. (If the device does not support files, specify only bits 7 and 6 and use zeros for bits 5–0.)

For example, the following device attributes byte would be used when opening a device for updating a file with the attributes display, variable, and sequential. The bit pattern (11000000) is equivalent to the hex value C0.

1	1	0	0	0	0	0	0
7	6	5	4	3	2	1	0

← least significant bit

This section provides additional information for using the Read and Write commands.

---

### Parameters in the PAB

The PAB for an Open command should include the following parameter values.

- ▶ **Buffer size**—Enter at least 0004. (The device always returns four bytes of information in response to the command.)
- ▶ **Data length**—Enter at least 0003. If you are specifying any device options, be sure to increase this value accordingly.

### After You Send the Command

After you send an Open command, the device compares the I/O buffer size you requested in your data with its own capabilities. It then responds to the command and returns four bytes of information to the data buffer, writing over the data you sent.

- ▶ **Accepted I/O buffer size (2 bytes)**
- ▶ **Record number (2 bytes)**

The accepted I/O buffer size is the device's response to your requested size. If the size you requested is acceptable, the device returns that value as confirmation. If you requested 0000, the device returns its default size. You can then use that value in the buffer-size field of PABs for other commands.

The record number specifies the current record in the device. (This is used only by devices that allow relative files; other devices always return 0000.)

If a condition such as a low battery or an unacceptable buffer size causes the device to return an error, the device is not opened. If the device was already open, it returns I/O error 5 (already open). In this case, the device remains open but ignores the parameters and data you sent.

## The Read and Write Commands

---

This section provides additional information for using the Read and Write commands.

---

### Read Command

The Read command, code 03, reads the next record in an open device and then stores that data in the calculator's data buffer.

When you send a Read command, pay particular attention to two of the parameters in the PAB.

- ▶ **Record number**—If you are using a device that allows you to access relative files, be sure to include the correct record number. (If you are performing successive read operations, you must increment the record number each time.)
- ▶ **Data length**—Enter 0000 (because you are not sending any data to the device).

After the Read operation is complete, the data length field of the PAB contains the number of bytes received from the device.

### Write Command

The Write command, code 04, instructs the calculator to send the information in the data buffer to the device.

Before you send a Write command, be sure to place the applicable information into the data buffer. You should also pay particular attention to two of the parameters in the PAB.

- ▶ **Record number**—If you are using a device that allows you to access relative files, be sure to include the correct record number. If you are performing successive Write operations, you must increment the record number each time. For Write commands to a device where records do not apply, such as a printer, you can use 0000.
- ▶ **Data length**—Enter the number of bytes of data you are sending to the device.

After you send an I/O command to a device, the device returns a status code which is stored in the returned status byte of the PAB and in the numeric display register. I/O errors do not halt a running program unless you have set the halt-on-error flag (flag 15). This allows the program to test the numeric display register and handle the error.

## Commonly-Used Error Codes

The following table lists the most commonly-used I/O error codes and gives a brief description of their meaning. Refer to your device manual for the specific error codes that apply to your device.

Error Code		Meaning
Decimal	Hex	
0	00	Operation completed successfully
1	01	Device option error
2	02	Attribute error
3	03	File/device not found
4	04	File/device not open
5	05	File/device already open
6	06	Device error (usually a hardware malfunction)
8	08	Data/file too long
9	09	Write/protect error
12	0C	Buffer size error
13	0D	Unsupported command error
25	19	Low batteries in the device
27	1B	Bus error (check the cable connections)
255	FF	Bus time-out error (device is not attached or did not respond within a specific time period)

In most cases, you need to perform an I/O operation from within a program. This section describes the steps involved in using CIO in a program.

## General Program Sequence

The steps generally required for a program that performs an I/O operation are:

1. Build the PAB and, if necessary, load the data buffer.
2. Execute the CIO function to send the command.
3. Test the returned status and, if applicable, read the data from the data buffer.

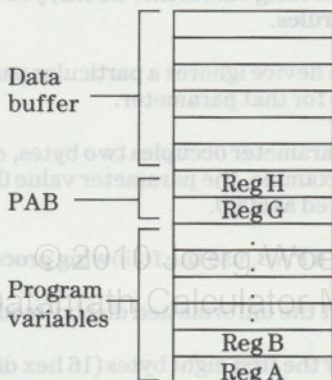
## Choosing Data Registers

Before writing the program, you should decide which data registers you want to use for the PAB, the data buffer, and any variables needed by the program.

- The PAB requires two consecutive data registers. Because the program uses only one PAB at a time, you can build the PAB for each command using the same two registers.
- The data buffer must be large enough to store the largest piece of information that you want to send to or receive from the device. For convenience, you should begin the buffer at the beginning of a register.
- The variables in the program can use any data registers that are not used by the PAB or the data buffer.

# **Choosing Data Registers (Continued)**

Although you can use any data registers for the information used by the program, you may want to use a sequence of registers. One possible way of arranging the information is shown below.



(continued)

### Building the PAB

You must begin a PAB in the first byte of a register. Because a PAB contains 12 bytes of information, you need to store it in two consecutive registers.

When entering values into a PAB, you should follow two general rules.

- ▶ If the device ignores a particular parameter, enter zeros for that parameter.
- ▶ If a parameter occupies two bytes, enter the LSB first. For example, the parameter value 0050 should be entered as 5000.

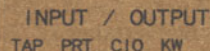
To build a PAB, use the following procedure.

1. Select the unformatted display mode.
2. Enter the first eight bytes (16 hex digits) of the PAB into the display.
3. Use the **[STO]** key to store the first eight bytes.
4. Enter the remaining four bytes into the display. (The last four bytes of the second register are ignored.)
5. Use the **[STO]** key to store the last four bytes of the PAB in the next consecutive register.

## Using the CIO Function

After building the PAB and storing any applicable information in the data buffer, you can then use the CIO function to send your I/O command to the device.

1. Press **[I/O]** to display the **INPUT/OUTPUT** menu.



2. Press **<CIO>**.
3. Enter the register number that contains the first eight bytes of the PAB.

The calculator then sends the I/O command to the device.

## Testing the Returned Status

There are several ways you can determine the success or failure of an I/O command after sending it. The method depends on whether you issued the command from the keyboard or from within a program.

- When you use CIO from the keyboard, any I/O error causes the calculator to display an error message specifying the error code.
- When you use CIO from within a program, no error message is displayed unless you halt or pause the program. The calculator does, however, place the returned status byte in the numeric display register. Your program can read this value to determine if an error occurred.
- Regardless of whether you use CIO from the keyboard or from within a program, your PAB will contain the returned status byte. You can use the RCB function to recall the byte. (Keep in mind that you can use the RCB function only after removing system protection.)

## Example: Using CIO to Control a Peripheral

---

This section gives an example of using the CIO function to control the optional PC-324 printer. In actual use, you can control the printer directly from the keyboard; you are not required to use the CIO function. The program does, however, illustrate how to use the function.

---

### Setting Up the Program

The sample program prints the message **HELLO THERE** five times, using the double-spacing option of the PC-324. It uses subroutines to send the necessary I/O commands—Open, Close, and Write—to the printer.

The device number for the PC-324 printer is usually given as the decimal number 12. To enter this value into a PAB, the program uses the hexadecimal equivalent, which is 0C.

Before writing such a program, you must determine which registers you want to use for the PAB, the data buffer, and any variables needed by the program. (Because this program uses a loop, it needs one register for the loop counter.)

- ▶ Reg A (000)—Loop counter
- ▶ Reg B (001)—PAB (bytes 1 through 8)
- ▶ Reg C (002)—PAB (bytes 9 through 12)
- ▶ Reg D (003)—Beginning of the data buffer

The PAB requires the hexadecimal address of the first byte in the data buffer. By using the formula on page A-6, you can calculate that the first byte in data register 003 is at address 3FE7.

The example uses the following labels for the subroutines.

- ▶ LBL OP—Subroutine for Open command
- ▶ LBL CL—Subroutine for Close command
- ▶ LBL WR—Subroutine for Write command

**The Main Program** The main program segment calls the three subroutines listed on the next few pages. This segment sets the halt-on-error flag because the program does not do its own error checking.

**Note:** This program begins by closing the printer. This is required because the TI-95 automatically opens the PC-324 printer if the printer is connected when you turn on the calculator. Because the program closes the printer at both the beginning and the end, attempting to run the program a second time produces I/O error 4 (not open). Before the program can be run a second time, the calculator must be turned off and then on.

Program Mnemonics	Comments
SF 15	Sets halt-on-error flag
SBL CL	Close printer
SBL OP	Open printer
'HELLO THERE'	Alpha message
STA D	Store message in data buffer
5 STO A	Set loop counter at 5
LBL AA	
SBL WR	Write message
DSZ A	
GTL AA	
SBL CL	Close printer at end of loop
HLT	
(remainder of program)	

(continued)

### Selecting Options for the Open Command

Before you send an Open command, you need to know the following information in order to store it in the data buffer.

- Requested I/O buffer size—The PC-324 uses a default buffer size of 24 bytes, which is equivalent to the two-byte hex value 0018. (In a case where you prefer to let a device establish the maximum buffer size for you, use 0000 in this field and read the returned value after the Open command has executed.)
- Device attributes—Use the output access mode and, because the printer does not support files, use zeros for the remaining bits. The resulting byte value is 80 hex.
- Device options—The double-spacing option is specified by the character string L=D. The ASCII values for these three characters are 4C3D44.

For the Open command in this example, the data buffer should contain the following values.

Buffer	Byte #	Parameter
18	1	Requested buffer size (LSB first)
00	2	
80	3	Device attributes
4C	4	Device options (in order)
3D	5	
44	6	

(continued)

# Subroutine for the Open Command

For the Open command, the PAB contains the values shown below.

PAB	Byte #	Parameter
0C	1	Device number—0C
00	2	Open command—00
00	3	LUNO (ignored)
00	4	Record number (ignored)
00	5	
04	6	Buffer size—0004 (LSB first)
00	7	
06	8	Data length—0006 (LSB first)
00	9	
00	10	Status to be returned from device
E7	11	Buffer pointer—3FE7 (LSB first)
3F	12	

The following subroutine loads the data buffer, builds the PAB, and sends the Open command.

Program Mnemonics	Comments
:	
:	
LBL OP	
UNF	Selects unformatted mode
1800804C3D44	Contents of data buffer
STO D	
0C00000000040006	First 8 bytes of PAB
STO B	
0000E73F	Remaining 4 bytes of PAB
STO C	
DEC	Restores decimal mode
CIO B	Sends the command
RTN	
:	
:	

(continued)

## Example: Using CIO to Control a Peripheral (Cont.)

### Subroutine for the Close Command

For the Close command, the PAB contains the values shown below.

PAB	Byte #	Parameter
0C	1	Device number—0C
01	2	Close command—01
00	3	LUNO (ignored)
00	4	Record number (ignored)
00	5	
18	6	Buffer size—0018 (LSB first)
00	7	
00	8	Data length—0000 (LSB first)
00	9	
00	10	Status to be returned from device
E7	11	Buffer pointer—3FE7 (LSB first)
3F	12	

The following subroutine builds this PAB and sends the Close command.

Program Mnemonics	Comments
:	
:	
LBL CL	
UNF	Selects unformatted mode
0C0100000018	First 8 bytes of PAB
STO B	(0C01000000180000)
0000E73F	Remaining 4 bytes of PAB
STO C	
DEC	Restores decimal mode
CIO B	Sends the command
RTN	
:	
:	

### Subroutine for the Write Command

For the Write command, the PAB contains the values shown below.

Because the data consists of the 11 characters HELLO THERE, the program specifies 000B (the hex equivalent of 11) as the data length.

PAB	Byte #	Parameter
0C	1	Device number—0C
04	2	Write command—04
00	3	LUNO (ignored)
00	4	Record number (ignored)
00	5	
18	6	Buffer size—0018 (LSB first)
00	7	
0B	8	Data length—000B (LSB first)
00	9	
00	10	Status to be returned from device
E7	11	Buffer pointer—3FE7 (LSB first)
3F	12	

The following subroutine builds this PAB and sends the Write command. (The main program has already stored the message in the data buffer.)

Program Mnemonics	Comments
:	
:	
LBL WR	
UNF	Selects unformatted mode
0C0400000018000B	First 8 bytes of PAB
STO B	
0000E73F	Remaining 4 bytes of PAB
STO C	
DEC	Restores decimal mode
CIO B	Sends the command
RTN	
:	
:	

## Reading Keystrokes from a Program

---

The <KW> (key wait) selection of the INPUT/OUTPUT menu pauses your program to wait for the next key to be pressed. This function can be used only in a program; if used directly from the keyboard, it is ignored.

---

### Key Codes

Each key on the keyboard is assigned a specific code from 0 to 63 (128 to 191 if you press **2nd** before pressing the key). A table of these codes is listed in Appendix C.

**Note:** Do not confuse key codes with program instruction codes or character codes. For example, the key code for the **SIN** (alpha letter “A”) key is 16, but the program instruction code for SIN is 165 (A5 hex) and the character code for “A” is 65 (41 hex).

### The Key Wait Function

The key wait function lets your program detect which key has been pressed by placing the code of the key in the numeric display register. You can use this function only as program instruction. The calculator executes the key wait instruction as follows:

- ▶ First, the instruction waits for a key (or second function of a key) to be pressed.
- ▶ After a key is pressed, the instruction places the code for that key in the numeric display register.
- ▶ Program execution then continues.

### Using the Key Wait Function

To include the key wait function in a program:

1. Press **[V/O]** and select <KW> from the INPUT/OUTPUT menu.
2. If you are expecting a particular key, use one of the test instructions to compare the code in the numeric display register with the code you are expecting.

The tables in this appendix contain reference information you may need while using the programming features of the TI-84.

### Example Program

Write a program that waits for the **1**, **2**, or **3** key to be pressed, and then executes a specific routine depending on which key was pressed.

PC =	Program Mnemonics	Comments
0000	LBL ST	
0003	CLR	Clears calculator
0004	'ROUTINE (1-3)?'	Creates message
0018	KW	Waits for key
0019	STO A	Stores key code in A
0021	63 IF= A GTL AA	If "1," executes label AA
0028	57 IF= A GTL BB	If "2," executes label BB
0035	41 IF= A GTL CC	If "3," executes label CC
0042	GTL ST	Otherwise, repeats prompt
0045	LBL AA	
0048	'ONE' HLT	User pressed "1"
0052	LBL BB	
0055	'TWO' HLT	User pressed "2"
0059	LBL CC	
0062	'THREE' HLT	User pressed "3"

### Running the Example

Run the program and press either the **1**, **2**, or **3** key. The program displays the message that corresponds to the key and then halts.



## Appendix C: Reference Information

---

The tables in this appendix contain reference information you may need while using the programming features of the TI-95.

---

<b>Table of Contents</b>	Character Codes .....	C-2
	Program Codes .....	C-4
	Key Codes .....	C-6
	System Flags .....	C-7
	Instruction Mnemonics .....	C-10
	Useful Assembly-language Subroutines .....	C-14
	Summary of Field Instructions .....	C-16
	Memory Map .....	C-17
	System RAM Usage .....	C-18
	Index .....	C-21

© 2010 Joerg Woerner  
Datamath Calculator Museum

This table lists each character code and its corresponding displayable character, if any. The codes can be used with the CHR selection of the alpha mode to display a character or to store a code in the alpha register for controlling your printer. Your printer may not be able to print some of the characters you can display.

000	032	BLANK	064	Q	096	~
001	033	!	065	R	097	a
002	034	"	066	B	098	b
003	035	#	067	C	099	c
004	036	\$	068	D	100	d
005	037	%	069	E	101	e
006	038	&	070	F	102	f
007	039	'	071	G	103	g
008	040	<	072	H	104	h
009	041	>	073	I	105	i
010	042	*	074	J	106	j
011	043	+	075	K	107	k
012	044	,	076	L	108	l
013	045	-	077	M	109	m
014	046	.	078	N	110	n
015	047	/	079	O	111	o
016	048	0	080	P	112	p
017	049	1	081	Q	113	q
018	050	2	082	R	114	r
019	051	3	083	S	115	s
020	052	4	084	T	116	t
021	053	5	085	U	117	u
022	054	6	086	V	118	v
023	055	7	087	W	119	w
024	056	8	088	X	120	x
025	057	9	089	Y	121	y
026	058	:	090	Z	122	z
027	059	;	091	[	123	{
028	060	<	092	\	124	
029	061	=	093	]	125	}
030	062	>	094	^	126	~
031	063	?	095	_	127	←

**Note:** Codes 000 through 031 are control codes. The characters displayed by these codes may vary.

128	BLANK	160	BLANK	192	タ	224	×
129	BLANK	161	□	193	チ	225	÷
130	BLANK	162	「	194	ツ	226	×
131	BLANK	163	」	195	テ	227	×
132	BLANK	164	、	196	ト	228	×
133	BLANK	165	・	197	ナ	229	×
134	BLANK	166	ヲ	198	ニ	230	×
135	BLANK	167	ア	199	ヌ	231	×
136	BLANK	168	イ	200	ネ	232	×
137	BLANK	169	ウ	201	ノ	233	×
138	BLANK	170	エ	202	ハ	234	×
139	BLANK	171	オ	203	ヒ	235	×
140	BLANK	172	カ	204	フ	236	×
141	BLANK	173	ク	205	ヘ	237	×
142	BLANK	174	コ	206	ホ	238	×
143	BLANK	175	ッ	207	マ	239	×
144	BLANK	176	ー	208	ミ	240	×
145	BLANK	177	ア	209	ム	241	×
146	BLANK	178	イ	210	メ	242	×
147	BLANK	179	ウ	211	モ	243	×
148	BLANK	180	エ	212	ヤ	244	×
149	BLANK	181	オ	213	ユ	245	×
150	BLANK	182	カ	214	ヨ	246	×
151	BLANK	183	キ	215	ラ	247	×
152	BLANK	184	ク	216	リ	248	×
153	BLANK	185	ケ	217	ル	249	×
154	BLANK	186	コ	218	レ	250	×
155	BLANK	187	サ	219	ロ	251	×
156	BLANK	188	シ	220	ワ	252	×
157	BLANK	189	ス	221	ン	253	×
158	BLANK	190	セ	222	。	254	BLANK
159	BLANK	191	ソ	223	。	255	■

## Program Codes

This table lists, as hexadecimal numbers, the codes used by the TI-95 to represent functions and alpha characters in program memory. Codes 20 through 7E are standard ASCII (American Standard Code for Information Interchange) characters. The codes 80 through 8F represent numeric values, not characters.

Code	Function	Code	Character	Code	Character	Code	Character
00	NOP	20	space	40	@	60	`
01	CE	21	!	41	A	61	a
02	CLR	22	"	42	B	62	b
03	CMS	23	#	43	C	63	c
04	x~t	24	\$	44	D	64	d
05	INV	25	%	45	E	65	e
06	HYP	26	&	46	F	66	f
07	HLP	27	'	47	G	67	g
08	PAU	28	(	48	H	68	h
09	HLT	29	)	49	I	69	i
0A	BRK	2A	*	4A	J	6A	j
0B	Y/N	2B	+	4B	K	6B	k
0C	EE	2C	,	4C	L	6C	l
0D	+/-	2D	-	4D	M	6D	m
0E	.	2E	.	4E	N	6E	n
0F	ENG	2F	/	4F	O	6F	o
10	=	30	0	50	P	70	p
11	)	31	1	51	Q	71	q
12	(	32	2	52	R	72	r
13	SG+	33	3	53	S	73	s
14	+	34	4	54	T	74	t
15	-	35	5	55	U	75	u
16	*	36	6	56	V	76	v
17	/	37	7	57	W	77	w
18	y^x	38	8	58	X	78	x
19	CUB	39	9	59	Y	79	y
1A	QAD	3A	:	5A	Z	7A	z
1B	nPr	3B	;	5B	[	7B	{
1C	nCr	3C	<	5C	\	7C	
1D	LCM	3D	=	5D	]	7D	}
1E	PF	3E	>	5E	^	7E	~
1F	CD	3F	?	5F	_	7F	←

Code	Function	Code	Function	Code	Function	Code	Function
80	0	A0	1/x	C0	DMS	E0	RCL
81	1	A1	$x^2$	C1	P-R	E1	EXC
82	2	A2	SQR	C2	F-C	E2	STO
83	3	A3	LN	C3	G-L	E3	ST +
84	4	A4	LOG	C4	i-m	E4	ST -
85	5	A5	SIN	C5	f-M	E5	ST*
86	6	A6	COS	C6	#-K	E6	ST/
87	7	A7	TAN	C7	D-R	E7	DSZ
88	8	A8	x!	C8	D-G	E8	INC
89	9	A9	ABS	C9	R-G	E9	IF<
8A	A	AA	INT	CA	DEC	EA	IF=
8B	B	AB	FRC	CB	HEX	EB	IF>
8C	C	AC	SGN	CC	OCT	EC	MRG
8D	D	AD	RND	CD	UNF	ED	RCA
8E	E	AE	13d	CE	2sC	EE	STA
8F	F	AF	DRG	CF	RUN	EF	RD
90	F1	B0	FRQ	D0	not used	F0	WRT
91	F2	B1	CS1	D1	not used	F1	VFY
92	F3	B2	CS2	D2	not used	F2	CIO
93	F4	B3	s	D3	not used	F3	WID
94	F5	B4	y'	D4	not used	F4	COL
95	LR	B5	MN	D5	SF	F5	GTL
96	LP	B6	m-b	D6	RF	F6	SBL
97	LL	B7	r	D7	TF	F7	DFN
98	LS	B8	PI	D8	DEV	F8	IND
99	R#	B9	OLD	D9	SHW	F9	GTO
9A	ASM	BA	INS	DA	CHR	FA	SBR
9B	PAR	BB	DEL	DB	FIX	FB	DFA
9C	CAT	BC	RTN	DC	DF	FC	SBA
9D	CFG	BD	TRC	DD	NAM	FD	STB
9E	WB	BE	PRT	DE	PUT	FE	RCB
9F	KW	BF	ADV	DF	GET	FF	LBL

# Key Codes

This table lists the codes returned to the <KW> (key wait) instruction of the INPUT/OUTPUT menu. Information on using the key wait function is given in Appendix B. To obtain the code for the second function of a key, such as **2nd** [DEL], add 128 to the code shown for the primary function of the key.

<b>OFF</b> 0	<b>EE</b> 1	<b>F1</b> 2	<b>F2</b> 3	<b>F3</b> 4	<b>F4</b> 5	<b>F5</b> 6	<b>(</b> 7
<b>BREAK</b> 8	<b>HALT</b> 9	<b><math>\Sigma+</math></b> 10	<b><math>x \sim t</math></b> 11	<b>HYP</b> 12	<b>INCR</b> 13	<b>EXC</b> 14	<b>STO</b> 15
<b>SIN</b> 16	<b>COS</b> 17	<b>TAN</b> 18	<b>LN</b> 19	<b>LOG</b> 20	<b><math>x^2</math></b> 21	<b><math>\sqrt{x}</math></b> 22	<b>1/x</b> 23
<b>I/O</b> 24	<b>FILES</b> 25	<b>STAT</b> 26	<b>CONV</b> 27	<b>NUM</b> 28	<b>FLAGS</b> 29	<b>TESTS</b> 30	<b>FUNC</b> 31
<b>HELP</b> 32	<b>ALPHA</b> 33	<b>LEARN</b> 34	<b>OLD</b> 35	<b>RUN</b> 36	<b><math>\leftarrow</math></b> 37	<b><math>\rightarrow</math></b> 38	<b>CE</b> 39
<b>)</b> 40	<b>3</b> 41	<b>6</b> 42	<b>9</b> 43	<b>0</b> 44	<b>RCL</b> 45	<b><math>y^x</math></b> 46	<b>LIST</b> 47
<b><math>\div</math></b> 48	<b>+</b> 49	<b>-</b> 50	<b><math>\times</math></b> 51	<b><math>\cdot</math></b> 52	<b>INV</b> 53	<b>2nd</b> 54	<b>CLEAR</b> 55
<b>=</b> 56	<b>2</b> 57	<b>5</b> 58	<b>8</b> 59	<b><math>\pm/\mp</math></b> 60	<b>7</b> 61	<b>4</b> 62	<b>1</b> 63

This table lists the system flags that are of general interest. Use the table to determine the meaning of each system flag and the effects of setting or resetting the flag. For instructions on how to set, reset, or test flags, refer to Chapter 5 of this guide.

Flag Number	Meaning	Comments
24	0 = F1-F5 are not defined 1 = F1-F5 are defined	Flag 25 specifies which set of F1-F5 definitions to use.
25	0 = F1-F5 defined by system 1 = F1-F5 defined by user	Ignored unless flag 24 = 1.
33	0 = Not radian mode 1 = Radian mode	If flags 33 and 34 both = 0, then degree mode. Controls <b>RAD</b> indicator.
34	0 = Not grad mode 1 = Grad mode	Controls <b>GRAD</b> indicator.
39	0 = Not second function 1 = Next key is a second function	Controls <b>2nd</b> indicator.
40	0 = Not unformatted mode 1 = Unformatted mode	If flags 40, 41, and 42 all = 0, then decimal mode.
41	0 = Not octal mode 1 = Octal mode	Controls <b>OCT</b> indicator.
42	0 = Not hexadecimal mode 1 = Hexadecimal mode	Controls <b>HEX</b> indicator.
43	0 = Not engineering notation display mode 1 = Engineering notation display mode	
44	0 = Not scientific notation display mode 1 = Scientific notation display mode	
49	0 = System protected 1 = System unprotected ( <b>SYS</b> mode in effect)	Controls <b>SYS</b> indicator.

(continued)

Flag Number	Meaning	Comments
50	0 = Remove contents of display, t-register for <b>INV</b> <b>Σ+</b> 1 = Remove last point for <b>INV</b> <b>Σ+</b>	Reset automatically if next sequence is not <b>INV</b> <b>Σ+</b> .
51	0 = Two-variable statistics mode 1 = One-variable statistics mode	
52	0 = <b>INV</b> was not pressed before F1-F5 1 = <b>INV</b> was pressed before F1-F5	Flag 52 is a copy of the status of the INV flag when F1-F5 was pressed. You can test this flag for an INV <i>Fx</i> sequence.
53	0 = No error was detected 1 = An error was detected	Controls <b>ERROR</b> indicator.
54	0 = Not word-break mode 1 = Word-break mode	
55	0 = Fetch new key 1 = Fetch last key pressed	System tests before next key scan (either KW or regular keyboard operation).
56	0 = Do not scroll user alpha message 1 = Scroll user alpha message if more than 16 characters	Reset automatically each time a new alpha display is sent out.
61	0 = Do not display a system alpha message 1 = Display a system alpha message	If flags 61 and 62 both = 0, the numeric display is sent out.
62	0 = Do not display a user alpha message 1 = Display a user alpha message	

Flag Number	Meaning	Comments
63	0 = Uppercase lock 1 = Lowercase lock	Controls the <b>LC</b> indicator.
67	0 = Not fixed-decimal mode 1 = Fixed-decimal mode	If flag 67 = 1, then byte 21A6 contains the number of digits to show after the decimal point.
68	0 = Not trace mode 1 = Trace mode	
72	0 = Not two's complement mode 1 = Two's complement mode	
73	0 = Auto power-down enabled 1 = Auto power-down disabled	
74	0 = Specified printer not connected 1 = Specified printer connected	Set/reset after each print or advance.
75	0 = Peripheral battery OK 1 = Low battery in peripheral	Controls <b>P</b> indicator.
76	0 = TI-95 battery OK 1 = Low battery in TI-95	Controls <b>LOW</b> indicator.
77	0 = No I/O activity 1 = I/O activity on the I/O bus	Controls <b>I/O</b> indicator.

# Instruction Mnemonics

The table on this and the following pages contains an alphabetical listing of the mnemonics displayed for program instructions. For example, the mnemonic #-K represents the pounds-to-kilograms function and is generated by the keystrokes **CONV** <MET> <#-K>. The program codes in the table are included for use by advanced programmers.

Mnemonic	Meaning	Key Sequence	Code (Hex)
#-K	Pounds to kilograms	<b>CONV</b> <MET> <#-K>	C6
(	Open parenthesis	<b>(</b>	12
)	Close parenthesis	<b>)</b>	11
+	Add	<b>+</b>	14
-	Subtract	<b>-</b>	15
*	Multiply	<b>x</b>	16
/	Divide	<b>÷</b>	17
.	Decimal point	<b>.</b>	0E
+ / -	Change sign	<b>+/-</b>	0D
=	Equals function	<b>=</b>	10
1/x	Reciprocal	<b>1/x</b>	A0
2sC	2's complement	<b>CONV</b> <BAS> <2sC>	CE
0	Numeric 0	<b>0</b>	80
1	Numeric 1	<b>1</b>	81
2	Numeric 2	<b>2</b>	82
3	Numeric 3	<b>3</b>	83
4	Numeric 4	<b>4</b>	84
5	Numeric 5	<b>5</b>	85
6	Numeric 6	<b>6</b>	86
7	Numeric 7	<b>7</b>	87
8	Numeric 8	<b>8</b>	88
9	Numeric 9	<b>9</b>	89
13d	Show 13 digits	<b>2nd</b> [13d]	AE
A	Hex digit A	<b>2nd</b> [A <sub>H</sub> ]	8A
ABS	Absolute value	<b>NUM</b> <--> <ABS>	A9
ADV	Paper advance	<b>2nd</b> [ADV]	BF
ASM	Assemble program	<b>2nd</b> [ASM]	9A
B	Hex digit B	<b>2nd</b> [B <sub>H</sub> ]	8B
BRK	Break	<b>BREAK</b>	0A
C	Hex digit C	<b>2nd</b> [C <sub>H</sub> ]	8C
CAT	Catalog directory	<b>FILES</b> <CAT>	9C
CD	Clear directory	<b>FILES</b> <--> <CD>	1F
CE	Clear entry	<b>CE</b>	01
CFG	Clear flags	<b>FLAGS</b> <CLR>	9D
CHR	Character code	<b>ALPHA</b> <--> <CHR>	DA
CIO	Call I/O	<b>I/O</b> <CIO>	F2
CLR	Clear	<b>CLEAR</b>	02
CMS	Clear memories	<b>2nd</b> [CMS]	03
COL	Alpha column	<b>ALPHA</b> <COL>	F4

Mnemonic	Meaning	Key Sequence	Code (Hex)
COS	Cosine	<b>COS</b>	A6
CS1	Clear 1-variable Stat Regs	<b>STAT</b> <CLR> <CS1>	B1
CS2	Clear 2-variable Stat Regs	<b>STAT</b> <CLR> <CS2>	B2
CUB	Cubic equation	<b>FUNC</b> <CUB>	19
D	Hex digit D	<b>2nd</b> [ <b>D<sub>H</sub></b> ]	8D
D-G	Degrees to grads	<b>CONV</b> <ANG> <D-G>	C8
D-R	Degrees to radians	<b>CONV</b> <ANG> <D-R>	C7
DEC	Decimal mode	<b>CONV</b> <BAS> <DEC>	CA
DEL	Alpha delete	<b>ALPHA</b> <DEL>	BB
DEV	Printer device number	<b>I/O</b> <PRT> <DEV>	D8
DF	Delete file	<b>FILES</b> <DF>	DC
DFA	Define absolute	Available through ASM	FB
DFN	Define function key	<b>2nd</b> [ <b>DFN</b> ]	F7
DMS	Degrees, minutes, seconds	<b>CONV</b> <DMS>	C0
DRG	Degrees, radians, grads	<b>2nd</b> [ <b>DRG</b> ]	AF
DSZ	Decrement & skip if zero	<b>TESTS</b> <DSZ>	E7
E	Hex digit E	<b>2nd</b> [ <b>E<sub>H</sub></b> ]	8E
EE	Scientific notation	<b>EE</b>	0C
ENG	Engineering notation	<b>2nd</b> [ <b>ENG</b> ]	0F
EXC	Exchange register	<b>EXC</b>	E1
F	Hex digit F	<b>2nd</b> [ <b>F<sub>H</sub></b> ]	8F
F1	F1 key (after DFN)	<b>F1</b>	90
F2	F2 key (after DFN)	<b>F2</b>	91
F3	F3 key (after DFN)	<b>F3</b>	92
F4	F4 key (after DFN)	<b>F4</b>	93
F5	F5 key (after DFN)	<b>F5</b>	94
F-C	Fahrenheit to centigrade	<b>CONV</b> <MET> <F-C>	C2
f-M	Feet to meters	<b>CONV</b> <MET> <f-M>	C5
FIX	Fix decimal	<b>2nd</b> [ <b>FIX</b> ]	DB
FRC	Fractional portion	<b>NUM</b> <FRC>	AB
FRQ	Frequency	<b>STAT</b> <FRQ>	B0
G-L	Gallons to liters	<b>CONV</b> <MET> <G-L>	C3
GET	Load file	<b>FILES</b> <GET>	DF
GTL	Go to label	<b>2nd</b> [ <b>GTL</b> ]	F5
GTO	Go to address	<b>INV</b> <b>2nd</b> [ <b>GTL</b> ]	F9
HEX	Hexadecimal display	<b>CONV</b> <BAS> <HEX>	CB
HLP	Help	<b>HELP</b>	07
HLT	Halt program	<b>HALT</b>	09
HYP	Hyperbolic function	<b>HYP</b>	06

Mnemonic	Meaning	Key Sequence	Code (Hex)
i-m	Inches to millimeters	<b>CONV</b> <MET> <i-m>	C4
IF =	If equal to	<b>TESTS</b> <IF = >	EA
IF >	If greater than	<b>TESTS</b> <IF >>	EB
IF <	If less than	<b>TESTS</b> <IF <>	E9
INC	Increment register	<b>INCR</b>	E8
IND	Indirect	<b>2nd</b> [IND]	F8
INS	Alpha insert	<b>ALPHA</b> <INS>	BA
INT	Integer portion	<b>NUM</b> <INT>	AA
INV	Inverse function	<b>INV</b>	05
KW	Key wait	<b>I/O</b> <KW>	9F
LBL	Label	<b>2nd</b> [LBL]	FF
LCM	Least common multiple	<b>NUM</b> <--> <LCM>	1D
LL	List labels	<b>LIST</b> <LBL>	97
LN	Natural log	<b>LN</b>	A3
LOG	Common log	<b>LOG</b>	A4
LP	List program	<b>LIST</b> <PGM>	96
LR	List registers	<b>LIST</b> <REG>	95
LS	List status	<b>LIST</b> <ST>	98
m-b	Slope/intercept	<b>STAT</b> <--> <m-b>	B6
MN	Mean	<b>STAT</b> <MN>	B5
MRG	Alpha merge	<b>ALPHA</b> <MRG>	EC
NAM	Name cartridge	<b>FILES</b> <--> <NAM>	DD
nCr	Combinations	<b>2nd</b> [nCr]	1C
NOP	No operation	<b>2nd</b> [NOP]	00
nPr	Permutations	<b>2nd</b> [nPr]	1B
OCT	Octal display	<b>CONV</b> <BAS> <OCT>	CC
OLD	Restore display	<b>OLD</b>	B9
P-R	Polar to rectangular	<b>CONV</b> <P-R>	C1
PAR	Partition	<b>2nd</b> [PART]	9B
PAU	Pause	<b>2nd</b> [PAUSE]	08
PF	Prime factor	<b>NUM</b> <--> <PF>	1E
PI	$\pi$	<b>2nd</b> [ $\pi$ ]	B8
PRT	Print display	<b>2nd</b> [PRINT]	BE
PUT	Save file	<b>FILES</b> <PUT>	DE
QAD	Quadratic equation	<b>FUNC</b> <QAD>	1A
r	Correlation coefficient	<b>STAT</b> <--> <r>	B7
R-G	Radians to grads	<b>CONV</b> <ANG> <R-G>	C9
R#	Random number	<b>NUM</b> <R#>	99
RCA	Recall alpha	<b>ALPHA</b> <--> <RCA>	ED

Mnemonic	Meaning	Key Sequence	Code (Hex)
RCB	Recall byte	<b>FUNC</b> <SYS> <RCB>	FE
RCL	Recall register	<b>RCL</b>	E0
RD	Read tape file	<b>I/O</b> <TAP> <RD>	EF
RF	Reset flag	<b>FLAGS</b> <RF>	D6
RND	Round function	<b>NUM</b> <RND>	AD
RTN	Return	<b>2nd</b> [RTN]	BC
RUN	Run program file	<b>RUN</b>	CF
s	Standard deviation	<b>STAT</b> <s>	B3
SBA	Call assembly language	<b>FUNC</b> <SYS> <SBA>	FC
SBL	Call subroutine label	<b>2nd</b> [SBL]	F6
SBR	Call subroutine address	<b>INV</b> <b>2nd</b> [SBL]	FA
SF	Set flag	<b>FLAGS</b> <SF>	D5
SG +	$\Sigma +$ (sigma +) for statistics	<b><math>\Sigma +</math></b>	13
SGN	Signum function	<b>NUM</b> <--> <SGN>	AC
SHW	Show statistics data	<b>STAT</b> <--> <SHW> F1-F5	D9
SIN	Sine	<b>SIN</b>	A5
SQR	Square root	<b><math>\sqrt{x}</math></b>	A2
ST +	Register addition	<b>STO</b> <b>+</b>	E3
ST/	Register division	<b>STO</b> <b><math>\div</math></b>	E6
ST*	Register multiplication	<b>STO</b> <b>x</b>	E5
ST -	Register subtraction	<b>STO</b> <b>-</b>	E4
STA	Store alpha	<b>ALPHA</b> <--> <STA>	EE
STB	Store byte	<b>FUNC</b> <SYS> <STB>	FD
STO	Store register	<b>STO</b>	E2
TAN	Tangent	<b>TAN</b>	A7
TF	Test flag	<b>FLAGS</b> <TF>	D7
TRC	Trace mode	<b>2nd</b> [TRACE]	BD
UNF	Unformatted display	<b>CONV</b> <BAS> <UNF>	CD
VFY	Verify tape file	<b>I/O</b> <TAP> <VFY>	F1
WB	Word break	<b>I/O</b> <PRT> <WB>	9E
WID	Print width	<b>I/O</b> <PRT> <WID>	F3
WRT	Write tape file	<b>I/O</b> <TAP> <WRT>	F0
x^2	Square	<b><math>x^2</math></b>	A1
x!	Factorial	<b>2nd</b> [x!]	A8
x~t	Exchange t-register	<b><math>x \sim t</math></b>	04
y^x	Universal power	<b><math>y^x</math></b>	18
y'	Predicted Y	<b>STAT</b> <--> <y'>	B4
Y/N	YES/NO test	<b>TESTS</b> <Y/N>	0B

## Useful Assembly-language Subroutines

This table lists assembly-language subroutines that you may want to use in your programs.

SBA	Action
128	Performs an I/O operation to determine if the printer specified by <DEV> is connected to the calculator. (The default setting for <DEV> is 012, the device number for the PC-324 printer.) System flag 74 is set if the printer is connected. Flag 74 is reset if the printer is not connected. Flag 74 is also set or reset, as appropriate, each time a print or advance function is performed.
216	Performs the <LC> function, toggling the uppercase/lowercase status and indicator of the alpha mode.
21A	Performs the <b>2nd</b> function, toggling the second function status of the calculator. The <b>2nd</b> indicator is displayed when the calculator is in the second function condition.
21C	Stores in memory address 2107 the ASCII code of the last key that was pressed. If there is no ASCII code associated with the key, a zero is stored in address 2107. This subroutine can be used after a <KW> instruction to find the ASCII code of the key that was pressed.
226	Displays data, similar to a pause instruction, but without the one-second delay of the pause instruction.
256	Updates the indicator information in the display.
268	Delays program execution for approximately one second. The display is not updated.
27C	Waits until no keys are held down before permitting program execution to continue.
358	Performs the <b>OFF</b> function, turning off the calculator.
36A	Clears the alpha register and sets the cursor to column one.

This table lists, in alphabetical order, the calculator functions that have fields. (See page xiii of this guide for a description of the notational conventions used to represent the fields.) Except for **BL**, **SHW**, **STN**, and **DTA**, all functions with fields can be indirectly addressed in a program.

SBA	Action
396	Tests whether the cartridge port contains a cartridge. If there is no cartridge in the port, execution proceeds normally. If there is a cartridge in the port, the instruction following this subroutine call is skipped.
3B2	Performs the <b>2nd</b> <b>[F:CLR]</b> function, clearing the function key definitions from the display.
490	Performs the <b>ON</b> function. Executing this subroutine stops program execution, clears the subroutine return stack, sets the program counter to step 0000, calculates the memory checksum, and displays the message <b>MEM MAY BE LOST</b> . The memory message is displayed because the contents of system memory have changed since the power off checksum was calculated.
49F	Enables the cassette motor.
4A2	Disables the cassette motor.

**Note:** Using assembly-language subroutines not included on this list can produce unexpected results, including loss of data in memory.

## Summary of Field Instructions

This table lists, in alphabetic order, the calculator functions that have fields. (See page xiii of this guide for a description of the notational conventions used to represent the fields.) Except for LBL, SHW, DFN, and DFA, all functions with fields can be indirectly addressed in a program.

Function	Field	Function	Field
CHR	nnn	LBL	aa
CIO	nnn <sup>1</sup> or X	MRG	nnn <sup>1</sup> or X or =
COL	nn	NAM	aaa
DEV	nnn	PUT	aaa
DF	aaa	INV PUT	aaa
INV DF	aaa	RCA	nnn <sup>1</sup> or X or =
DFA <sup>2</sup>	Fx:aaa@nnnn	RCB	hhhh
DFN	Fx:aaa@aa	RCL	nnn <sup>1</sup> or X
DFN CLR		RD	aaa
DFN Fx CLR		RF	nn
DSZ	nnn <sup>1</sup> or X	SBA	hhh
INV DSZ	nnn <sup>1</sup> or X	SBL	aa
EXC	nnn <sup>1</sup> or X	INV SBL	nnnn
FIX	n	SF	nn
GET	aaa	SHW <sup>3</sup>	n
INV GET	aaa	ST*	nnn <sup>1</sup> or X
GTL	aa	ST +	nnn <sup>1</sup> or X
INV GTL	nnnn	ST -	nnn <sup>1</sup> or X
IF<	nnn <sup>1</sup> or X	ST/	nnn <sup>1</sup> or X
INV IF<	nnn <sup>1</sup> or X	STA	nnn <sup>1</sup> or X
IF=	nnn <sup>1</sup> or X	STB	hhhh
INV IF=	nnn <sup>1</sup> or X	STO	nnn <sup>1</sup> or X
IF>	nnn <sup>1</sup> or X	TF	nn
INV IF>	nnn <sup>1</sup> or X	INV TF	nn
INC	nnn <sup>1</sup> or X	VFY	aaa
INV INC	nnn <sup>1</sup> or X	WID	nn
IND <sup>4</sup>	nnn <sup>1</sup> or X	WRT	aaa

### Notes:

1. These numeric register addresses are four digits in system mode.
2. DFA cannot be keyed into a program. DFA instructions are generated by assembling a program.
3. SHW operates as a field instruction only in a program.
4. IND is used only after an instruction that requires a field.

# Memory Map

This table shows how the TI-95 memory is allocated.

Byte  
Address  
(hex)

0000 007F	System RAM (128 bytes)	
0080 1FFF	System reserved area	
2000 23DF	System RAM (992 bytes)	
23E0	File 1 File 2 File 3 . . Catalog	File space (5200 bytes)
382F		
3830	Step 0000 Step 0001 . . Step 0999	Program memory (1000 bytes)
3C17		User Memory (shown with default partitions)
3C18	Register 124 . . Register 001 (B) Register 000 (A)	Data registers (1000 bytes)
3FFF		
4000 BFFF	Cartridge port	(32K bytes)
C000 DFFF	System ROM (4 pages, 8K bytes each)	
E000 EFFF	System reserved area	
F000 FFFF	System ROM (4K bytes)	

This table lists the usage assigned to specific registers and addresses for the TI-95.

System Register	Byte Address (hex)	Contents
000-003	0000-001F	Temporary system usage
004	0020-0023	Temporary system usage
	0024	Current program code
	0025	Current key code
	0026-0027	Temporary system usage
005	0028-0029	User flags 00-15
	002A-002F	System flags 16-63
006	0030-0034	System flags 64-99
	0035	Temporary system usage
	0036	AOS stack counter
	0037	Temporary system usage
007	0038-003F	Numeric display register
008-013	0040-006F	Temporary system usage
014	0070-0071	Temporary system usage
	0072-0073	1st address in currently running program or LEARN
	0074-0075	End address in currently running program or LEARN
	0076-0077	PC in currently running program or LEARN
015	0078-007F	Reserved for use by application cartridges
016	2000-2007	1st AOS operand
017	2008-200F	2nd AOS operand
018	2010-2017	3rd AOS operand
019	2018-201F	4th AOS operand
020	2020-2027	5th AOS operand
021	2028-202F	6th AOS operand
022	2030-2037	7th AOS operand
023	2038-203F	8th AOS operand
024-026	2040-2057	Temporary system usage
027	2058-2059	If not expected values, memory will be cleared
	205A	User subroutine counter
	205B	Last accepted key code
	205C-205F	Temporary system usage
028	2060-2067	Operators for AOS operands

System Register	Byte Address (hex)	Contents
029-033	2068-208F	Temporary system usage
034	2090-2093	Temporary system usage
	2094	Error code
	2095-2097	Temporary system usage
035-036	2098-20A7	Temporary system usage
037	20A8-20AC	Temporary system usage
	20AD-20AF	Name of file currently running
038	20B0-20B1	Number of registers in partition
	20B2-20B3	Address of 1st byte in register area
	20B4-20B5	Address of 1st byte in program area
	20B6-20B7	Program area PC
039	20B8-20BF	Characters for user definitions of F1-F5
040	20C0-20C6	Characters for user definitions of F1-F5
	20C7	Flags and addresses for user definitions of F1-F5
041-043	20C8-20DF	Flags and addresses for user definitions of F1-F5
044	20E0-20E4	Flags and addresses for user definitions of F1-F5
	20E5-20E7	Temporary system usage
045-047	20E8-20FF	Temporary system usage
048	2100-2106	Temporary system usage
	2107	Result of SBA 21C
049-054	2108-2137	Subroutine return stack
055	2138	Cursor position for alpha register
	2139	Cursor position for system messages
	213A	Printer device-number
	213B	Printer width
	213C-213F	Temporary system usage
056-065	2140-218F	Alpha register
066-067	2190-219F	System message
068	21A0-21A5	Temporary system usage
	21A6	Value for fix decimal
	21A7	Temporary system usage
069	21A8-21AF	Temporary system usage

(continued)

System Register	Byte Address (hex)	Contents
070	21B0-21B7	Sum of Y's for statistics
071	21B8-21BF	Sum of Y^2's for statistics
072	21C0-21C7	N for statistics
073	21C8-21CF	Sum of X's for statistics
074	21D0-21D7	Sum of X^2's for statistics
075	21D8-21DF	Sum of X*Y's for statistics
076	21E0-21E7	Last X entered for statistics
077	21E8-21EF	Last Y entered for statistics
078	21F0-21F7	Frequency for statistics
079	21F8-21FF	t-register
080-101	2200-22AF	Temporary system usage
102	22B0-22B7	Flags and address for last BREAK command
103	22B8-22BA	Flags and address for last BREAK command
	22BB-22BF	Temporary system usage
104-108	22C0-22E7	Temporary system usage
109-110	22E8-22F7	Printer PAB
111	22F8-22FF	Temporary system usage
112-113	2300-230F	Random number seed
114-139	2310-23DF	Temporary system usage

(continued)

Topics listed here are related to the programming features of the TI-95. Topics related to the scientific-calculator functions are listed in the *TI-95 User's Guide*.

- A**
- Address,
    - calculating user memory, A-6
    - determining, 4-10
    - direct, 6-2
    - indirect, 6-2
  - Allocating memory, see partitioning
  - Alpha characters,
    - deleting, 3-11, 3-18
    - inserting, 3-11, 3-18
    - replacing, 3-10
  - Alpha cursor, positioning, 3-10, 3-18, 3-19
  - Alpha delete as an instruction, 3-18
  - Alpha insert as an instruction, 3-18
  - Alpha insert, cancelling, 3-11
  - Alpha merge, 3-12, 3-19
  - Alpha message,
    - appending, 3-14
    - clearing, 3-11
    - editing, 3-10
    - entering, 3-5
    - recalling, 3-9, 3-19, 3-20
    - rescrolling, 3-5
    - restoring, 4-21
    - scrolling, 3-5
    - stopping a scrolling, 3-5
    - storing, 3-8, 3-9, 3-20
  - Alpha mode, 3-2, 3-17
  - Alpha mode,
    - activating, 3-4
    - case of letters, 3-6
    - keyboard, 3-3
    - menu, 3-4
  - Alpha register, 3-17
  - Appending an alpha message, 3-14
  - ASCII codes, C-4
  - Assembling a program, 4-25, 4-29
  - Assembly-language subroutines,
    - table, C-14
    - using, A-15
- B**
- Break, 2-15
  - Break as an instruction, 2-10
- Byte,**
- calculating address of, A-6
  - storing and recalling, A-6
- C**
- Calculating address in user memory, A-6
  - Call, 4-11
  - Cassette interface, 9-6
  - Cassette recorder,
    - connecting, 9-6
    - operating prompts, 9-11, 9-14
    - selecting, 9-3
    - setting volume, 9-7
    - using, 9-4
  - Cassette tape, protection of, 9-5
  - Catalog listing, 8-18, 8-28, 8-31
  - Character codes, 3-20
  - Character codes,
    - table of, C-2
    - using, 3-7
  - CIO,
    - overview, B-3
    - use in a program, B-12
  - Clearing,
    - alpha message, 3-11
    - directory, 8-21, 8-28, 8-32
    - files, 8-20, 8-28, 8-32
    - function keys, 4-22, 4-23, 4-28, 4-29
    - program memory, 1-6, 1-18
  - Codes,
    - character, C-2
    - key, C-6
    - program, C-4
  - Command codes for CIO, B-6
  - Comparison tests, 5-4, 5-25
  - Connecting a cassette recorder, 9-6
  - Constant Memory cartridge, naming, 8-5, 8-25, 8-27, 8-31
  - CUB, 2-9
  - Cursor,
    - moving in alpha mode, 3-10
    - moving in learn mode, 1-12
    - positioning in learn mode, 1-6

- D**
- Data buffer for CIO, B-2
  - Data files, 8-24
  - Data files,
    - loading, 8-16, 8-27, 8-30
    - saving, 8-12, 8-26, 8-29
  - Data registers, partitioning, 7-7
  - Decrement, rules with DSZ, 5-7
  - Defining function keys, 4-16, 4-28
  - Deleting,
    - alpha characters, 3-11, 3-18
    - files, 8-20, 8-21, 8-28, 8-32
    - instructions, 1-15, 1-19
  - DFA instruction, 4-25, 4-29
  - Direct addressing, 6-2
  - Directory, 8-3
  - Directory,
    - catalog of, 8-18, 8-28, 8-31
    - clearing, 8-21, 8-28, 8-32
    - names, 8-3
  - Disassembling a program, 4-25, 4-29
  - Display with PAU, 2-13, 2-15
- E**
- Editing,
    - alpha message, 3-10
    - program, 1-14
  - Error codes for CIO, B-11
  - Error conditions, 1-5, 2-9, 4-28, 5-13, 6-11, 7-10, 7-15, 9-15, 9-23, 9-25, B-9, B-11, B-15
- F**
- Field instructions, table of, C-16
  - Fields, xiii
  - Fields,
    - indirect, 6-3, 6-14
    - storing functions with, 2-3
    - unprotected mode, A-11
  - File operations, from the keyboard, 8-25
  - File operations, tape, 9-20
  - File space, partitioning, 7-6
  - Files,
    - clear directory, 8-21, 8-28, 8-32
    - data, 8-24
    - deleting, 8-20, 8-28, 8-32
    - reading tape, 9-12, 9-22, 9-24
    - rules for naming, 8-2, 8-23, 9-2
    - tape, 9-2
    - tape operating prompts, 9-11, 9-14
    - tape operations from keyboard, 9-21
    - tape operations in a program, 9-18, 9-21
    - verifying tape, 9-12, 9-23, 9-25
    - writing tape, 9-9, 9-22, 9-24
  - Flag, halt-on-error, 5-13, 5-27, B-11
  - Flags,
    - setting, resetting, testing, 5-13, 5-27
    - system, A-2, C-7, C-8, C-9
    - table of system, C-7
    - user, 5-13, 5-27
  - Formatted mode, A-8
  - Function keys,
    - clearing, 4-22, 4-23, 4-28, 4-29
    - defining, 4-16, 4-28
  - Functions as program instructions, 2-2
  - Functions, keyboard, 2-9
- G-H**
- <GO>, 2-10
  - Halting a program, 1-20, 2-15
  - Halt-on-error flag, 5-13, 5-27, B-11
  - Help function, 2-9, 10-2, A-5

- I**
- Indicators,
    - ALPHA**, 3-4
    - INS**, 1-14, 3-18
    - I/O**, 9-11, 9-14
    - LC**, 3-6, 3-20
    - RUN**, 1-10
    - SYS**, A-4
  - Indirect addressing, 6-2, 6-14
  - Indirect addressing,
    - advantages of, 6-5
    - field information, 6-14, C-16
    - restrictions, 6-3
  - Infinite loop, 4-9
  - Input, numeric, 2-10
  - Input/Output with CIO, B-2
  - Inserting,
    - alpha characters, 3-11, 3-18
    - instructions, 1-14, 1-19
  - Instruction, 1-4
  - Instruction,
    - deleting, 1-15, 1-19
    - determining address of, 4-10
    - inserting, 1-14, 1-19
    - labeling, 4-5
    - replacing, 1-16
  - Instruction mnemonics, table of, C-10
  - Internal representation of numbers, A-13
  - Interrupting a program for input, 2-10
  - Inverse functions, storing, 2-2
- K**
- Key codes, table of, C-6
  - Key wait function, B-22
- L**
- Labels, 4-5, 4-26
  - Labels,
    - advantages of, 4-6
    - assembling and disassembling, 4-25, 4-29
    - duplication of, 4-5
    - listing, 4-24, 4-26
    - using, 10-4
  - Learn mode, 1-18
  - Learn mode,
    - activating, 1-6
    - definition, 1-6
    - exiting, 1-7
    - menu, 1-6
    - restoring PC, 2-6, 2-15
  - Levels of subroutines, 4-12
  - Listing,
    - file catalog, 8-18, 8-28, 8-31, 8-32
    - labels, 4-24, 4-26
    - program, 1-12, 1-21
    - status, 2-9, 5-13
  - Listing, controlling a, 1-12
  - Listing, stopping a, 1-13
  - Loading,
    - data file, 8-16, 8-27, 8-30
    - program file, 8-10, 8-27, 8-30
  - Loop, 4-8
  - Loop,
    - exiting with a test, 5-22
    - infinite, 4-9
    - stopping a, 4-9
  - Lowercase lock, 3-6, 3-20
- M**
- Memory,
    - system, C-18
    - user, 7-2
  - Memory conflicts, avoiding, 10-4
  - Memory map, C-17
  - Memory partitioning,
    - checking, 7-8, 7-14
    - effects of, 7-6
    - from keyboard, 7-9, 7-14
    - within a program, 7-12, 7-16

### M (Continued)

- Menus,
  - clearing, 4-22
  - restoring, 4-21, 4-29
  - user-defined, 4-3, 4-18
- Merge number with message, 3-12, 3-19
- Mnemonic, 1-4
- Mnemonics, table of, C-10
- Mode,
  - alpha, 3-2
  - insert, 1-14
  - learn, 1-6, 1-18
  - protected/unprotected, A-2, A-3, A-11
  - unformatted, A-8

### N

- Naming files, 9-2
- NOP instruction, 1-6, 1-19
- Notational conventions, xiii
- Numbers, internal representation, A-13

### O

- OLD, 3-17, 4-21, 4-29

### P

- Partitioning,
  - checking, 7-8, 7-14
  - effects of, 7-6
  - from the keyboard, 7-9, 7-14
  - within a program, 7-12, 7-16
- Partitions, default, 7-3
- Pausing a program, 2-13, 2-15
- Peripheral access block (PAB), B-2
- Peripheral access block,
  - building a, B-14
  - structure of, B-4
- Planning a program, 10-6
- Pointer register, 6-3, 6-14
- Printer control codes, 3-7

- Program,
  - assembly and disassembly, 4-25, 4-29
  - editing, 1-14
  - entering, 1-7
  - file operations in, 8-22, 8-25
  - initialization, 10-2
  - planning, 10-6
  - running, x, 1-10, 1-20
  - saving as a file, 8-6, 8-24
  - stopping, 1-5, 1-19
  - table of mnemonics, C-10
  - tape operations, 9-18, 9-21
  - testing, 10-9

- Program codes, table of, C-4

- Program counter, 1-5

- Program counter,
  - restoring, 2-6, 2-15
  - setting, 4-26
  - using, 4-10

- Program file,
  - execute routine, 8-34
  - listing, 1-12, 1-21
  - loading, 8-10, 8-27, 8-30
  - running, 8-8, 8-33, 8-34
  - saving, 8-26, 8-29

- Program instructions with fields, C-16

- Program memory,
  - clearing, 1-6, 1-18
  - partitioning, 7-6

- Program step, 1-3

- Program step, changing number of, 7-9, 7-12, 7-14

- Programming, x

- Programming considerations, 10-3

- Programming features, xi

- Protected mode, A-2

- Protection, removing system, A-4

### Q

- QAD, 2-9

## R

- Reading tape files, 9-12, 9-22, 9-24
- Recalling alpha messages, 3-9, 3-14, 3-19, 3-20
- Recalling bytes, A-6
- Register, pointer, 6-3, 6-14
- Registers,
  - system, A-2, A-11, C-18, C-20
  - use by QAD and CUB, 2-9
- Replacing alpha characters, 3-10
- Replacing an instruction, 1-16
- Representation of numbers, A-13
- Restoring,
  - alpha message, 3-17, 4-21
  - program counter, 2-6, 2-15
  - user-defined menu, 4-21, 4-29
- Return from subroutine, 4-11, 4-27
- Return stack, subroutine, 4-28
- Running a program, 1-10, 1-20
- Running a program file, 8-8, 8-33, 8-34

## S

- Saving a data file, 8-12, 8-26, 8-29
- Saving a program file, 8-6, 8-24, 8-26, 8-29
- Scrolling,
  - in alpha mode, 3-5
  - stopping, 3-5
- Selecting a cassette recorder, 9-3
- Selecting tapes, 9-3
- Setting cassette recorder volume, 9-7
- Setting the program counter, 4-26
- Sign digit, A-13
- Status, listing, 2-9
- Step, 1-3
- Stopping a program, options, 2-15
- Storing alpha messages, 3-9, 3-20
- Storing functions,
  - bytes, A-6
  - inverse, 2-2
  - system-menu, 2-6
  - with fields, 2-3
- Subroutine, 4-11

## Subroutine,

- assembly-language, A-15
  - avoiding problems with, 4-14
  - levels of, 4-12
  - return from, 4-11
  - return stack, 4-28
  - table of assembly-language, C-14
- System flags, table of, C-7
- System memory usage, C-18
- System-menu functions, storing, 2-6
- System protection, A-4
- System registers, A-2, A-11, C-18, C-20

## T

- Tapes, selecting, 9-3
- Test instructions,
  - comparison, 5-4, 5-25
  - DSZ, 5-7, 5-26
  - flag, 5-13, 5-27
  - Y/N, 5-11, 5-26
- Testing a program, 10-9
- Tests, 5-2
- Tests,
  - action of, 5-2
  - types of, 5-2
- Transfer instruction,
  - GTL, 4-8, 4-26
  - GTO, 4-10, 4-26
  - SBL, 4-11, 4-13, 4-27
  - SBR, 4-13, 4-27
  - using with RUN, 8-34
  - using with tests, 5-16

## U

- Unassembling a program, 4-25, 4-29
- Unformatted mode, A-8
- Unprotected mode, A-2, A-11
- Unprotected mode, precautions, A-3
- Unprotecting the system, A-4, A-11
- User memory, organization of, 7-2
- User-defined menus, 4-3, 4-18
- User-defined menus,
  - clearing, 4-22, 4-28
  - restoring, 4-21, 4-29
- Using a cassette recorder, 9-4

## V-W

Verifying tape files, 9-12, 9-23, 9-25

Writing tape files, 9-9, 9-22, 9-24

→ key in alpha mode, 3-5, 3-18

→ key in learn mode, 1-18

→ key to control listing, 1-18

→ key in alpha mode, 3-5, 3-18

→ key in learn mode, 1-19



