# 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.

**Table of Contents**

## Introduction

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.<br><br>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.<br><br>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 $\boxed{2nd}$ [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^x 3 * PI | |
| | * 4/3 = | Calculates volume |
| 0022 | 'VOL=' | Creates message |
| 0026 | COL 16 MRG = | Positions cursor and merges |
| 0030 | HLT | Stops program |

## Using Program Labels

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:

$\boxed{2nd}$ [LBL] *aa*

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

After you press $\boxed{2nd}$ [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 | $\boxed{2nd}$ [LBL] $\boxed{2nd}$ F $\boxed{2nd}$ X |
| LBL 10 | $\boxed{2nd}$ [LBL] 10 |
| LBL Z1 | $\boxed{2nd}$ [LBL] Z1 |
| LBL Aa | $\boxed{2nd}$ [LBL] A $\boxed{2nd}$ A |

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.

## Why Use Labels? (Continued)

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

| Before Insertion | | After Insertion | |
|---|---|---|---|
| PC = | Mnemonics | PC = | Mnemonics |
| 0130 | LBL AA | 0130 | RCL B |
| 0133 | * PI = | 0132 | LBL AA |
| 0136 | PAU | 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.

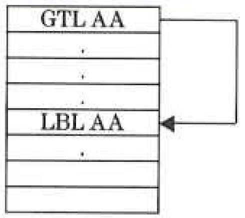
To enter the GTL instruction, use the key sequence:

2nd [GTL] *aa*

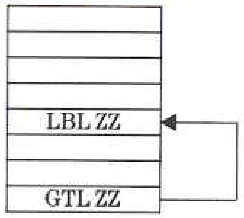
where *aa* corresponds to the label in the program.

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 = 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 | CLEAR | 0. |
| Run the program | RUN ⟨PGM⟩ | 2. |
| | | 4. |
| Stop the program | HALT | 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 $\boxed{\text{LEARN}}$ and use ‹1st›, ‹PC›, or ‹END› to enter the learn mode at the point nearest the instruction.

2. Use the $\boxed{\rightarrow}$ and $\boxed{\leftarrow}$ 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

$\boxed{\text{INV}}$ $\boxed{\text{2nd}}$ [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.

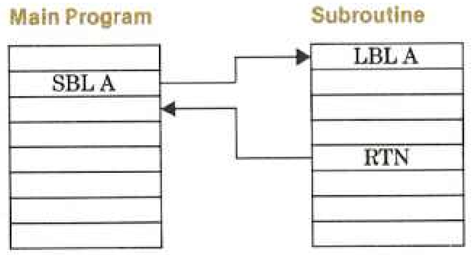
To include a subroutine in a program, use the following procedure.

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 $\boxed{\text{2nd}}$ [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.

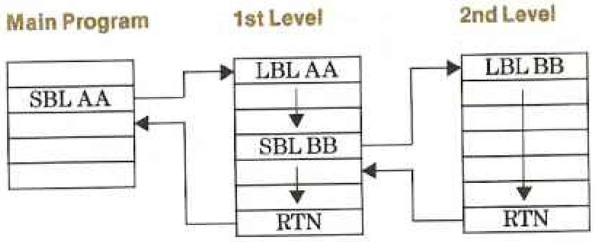


(continued)

## 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.)

## 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.

## 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 | [RUN] 〈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.

### 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 [F1] key | [2nd] [DFN] | DFN |
| | [F1] | 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.

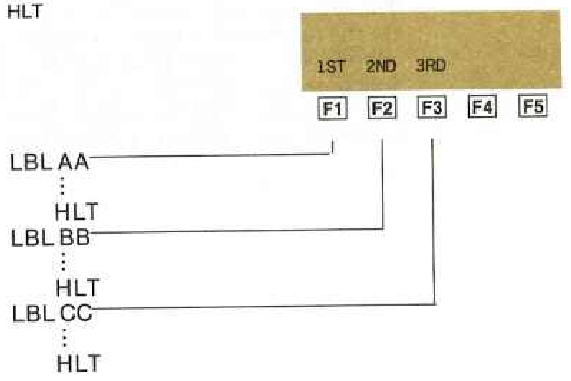
### Building a Menu

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 | [RUN] ⟨PGM⟩ | ROOTS<br>3RD 4TH 5TH |
| Enter a number | 8 | 8 |
| Calculate 3rd root | ⟨3RD⟩ | 2. |
| Enter a number | 3125 | 3125 |
| Calculate 5th root | ⟨5TH⟩ | 5. |

(continued)

## Creating a Function-Key Menu (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 | [RUN] ⟨PGM⟩ | ENTER SIDES<br>a b | <br>CAL |
| Enter side a | 30 ⟨a⟩ | | 30. |
| Enter side b | 40 ⟨b⟩ | | 40. |
| Calculate hypotenuse | ⟨CAL⟩ | HYP= | 50. |

## Restoring a User-Defined Menu

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.

## Clearing Function-Key Definitions

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

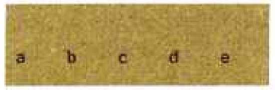
### Why Clear the Function Keys?

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

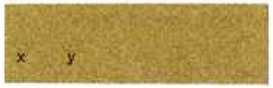
- You may want to define a function key for a limited time. By clearing the function key, you erase the function description from the display and cancel the function definition.
- If you use more than one menu in the program, you may need to clear some portion of the menu.

### Example

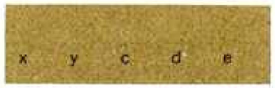
For example, suppose you want to display the following menus at different points in a program. The first menu has five options.



The second menu has two options.



When defining the second menu, you cannot redefine just the $\boxed{\text{F1}}$ and $\boxed{\text{F2}}$ keys. You must also clear $\boxed{\text{F3}}$, $\boxed{\text{F4}}$, and $\boxed{\text{F5}}$. Otherwise, the second menu would display:



### 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 $\boxed{\text{F1}}$ and $\boxed{\text{F2}}$.
- 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

  $\boxed{\text{2nd}}$ [DFN] $\boxed{\text{CLEAR}}$

  The mnemonic for $\boxed{\text{2nd}}$ [DFN] $\boxed{\text{CLEAR}}$ is DFN CLR.

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

  $\boxed{\text{2nd}}$ [DFN] *Fx* $\boxed{\text{CLEAR}}$

  where *Fx* represents the function key ($\boxed{\text{F1}}$–$\boxed{\text{F5}}$) that you want to clear.

  The mnemonic for $\boxed{\text{2nd}}$ [DFN] *Fx* $\boxed{\text{CLEAR}}$ 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.

## Speeding Up Program Execution

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

```
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
```

After assembly

```
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.