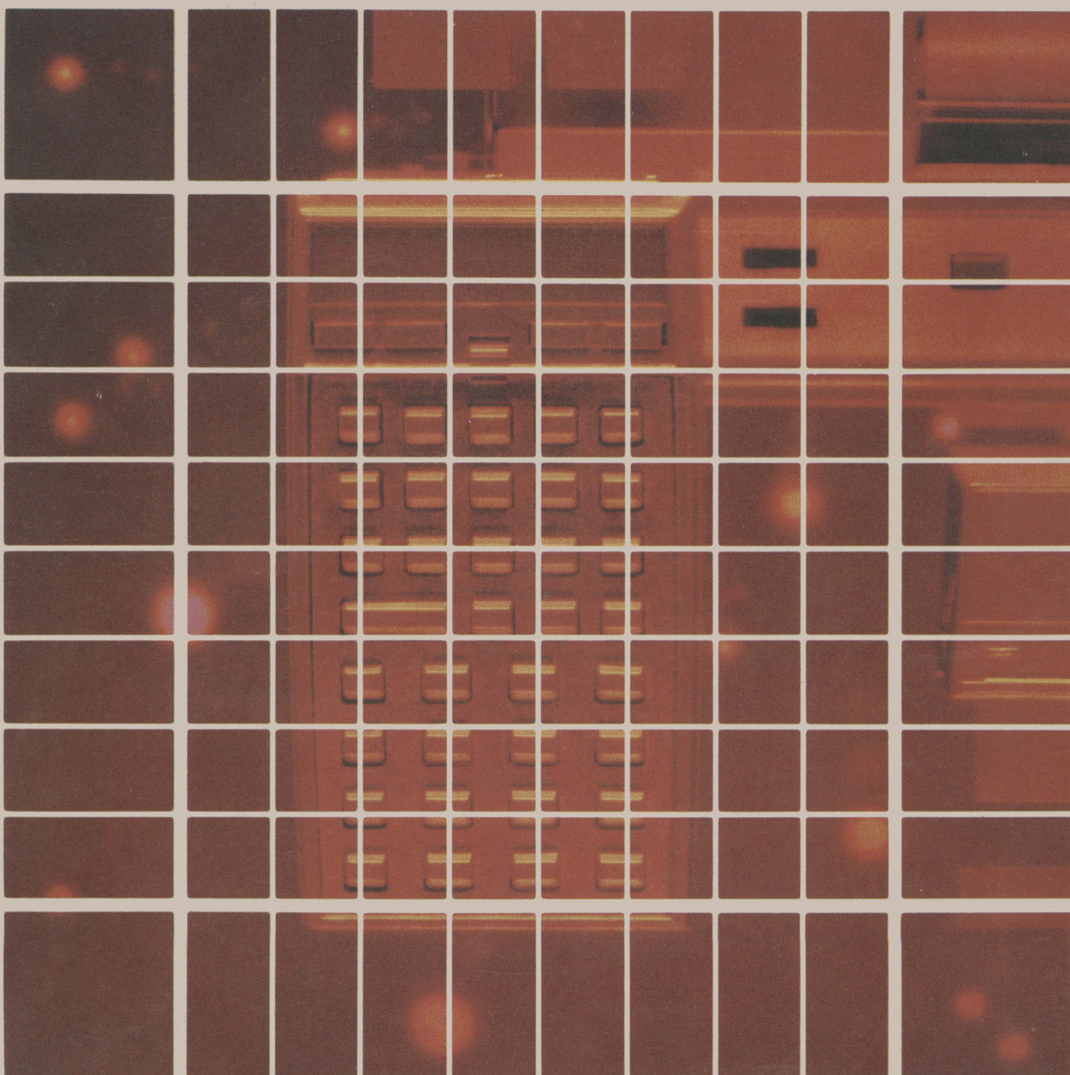
HEWLETT-PACKARD

HP-41CV

OPERATION IN DETAIL



Summary of Conventions Used in This Manual

Notation (Example)	Description
[STO]	*Black keybox.* Primary keyboard function.
[10^x]	*Gold keybox.* Shifted keyboard function. Press and release the shift key ( ) first. These can be on the Normal or Alpha keyboard.
[END]	*Blue keybox.* Nonkeyboard function. For Alpha execution: use [XEQ] followed by the Alpha name spelled out on the Alpha keyboard. For User-key execution: assign the function to the User keyboard.
ABC	*Blue letters.* Alpha characters.
123	*Gold digits or characters.* Shifted Alpha characters.
[X] [·] [T]	*Black letters in keyboxes.* These are special functions, *not* Alpha characters, and are active only in special circumstances.
parameter	*The type of parameter* required for a function.

For a full description, refer to "How This Manual Represents Keystrokes," page 14 in the *HP-41CV Owner's Manual.*

HEWLETT PACKARD

HP-41CV
Operation in Detail

April 1986

00041-90531 Rev. B

Printed in U.S.A.

Printing History

Edition 1 .... July 1984

Notice

Hewlett-Packard Company makes no express or implied warranty with regard to the keystroke procedures and program material offered or their merchantability or their fitness for any particular purpose. The keystroke procedures and program material are made available solely on an "as is" basis, and the entire risk as to their quality and performance is with the user. Should the keystroke procedures or program material prove defective, the user (and not Hewlett-Packard Company nor any other party) shall bear the entire cost of all necessary correction and all incidental or consequential damages. Hewlett-Packard Company shall not be liable for any incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of the keystroke procedures or program material.

Introducing *HP-41CV Operation in Detail*

This is a companion volume to the *HP-41CV Owner's Manual*, which covered "Basic Operation." This manual is an advanced, detailed examination of *all* aspects of the HP-41CV.

The organization of this manual emphasizes *reference* information and *completeness* of information.

- Part I is "Fundamentals in Detail."
- Part II is "Programming in Detail."
- There is a comprehensive summary of all the functions in the Function Tables (just in front of the subject index).
- The Function Index is listed inside the back cover.
- There are appendices about error conditions, null characters, printer operation, and peripherals devices available for the HP-41.

HP-41CV Operation in Detail
Contents

Part I: Fundamentals in Detail

Part II: Programming in Detail

Appendices

List of Diagrams and Tables

Part I:
Fundamentals in Detail

Section 1

The Keyboard and Display

Contents

The Toggle Keys

Just below the display are four toggle keys labeled [ON], [USER], [PRGM], and [ALPHA]. They control how the computer interprets the other keys. The toggle keys are so named because of their dual action: when you press one, it gives a particular interpretation to the keyboard which generally continues until you press the same toggle key again, returning the keyboard to its previous state.

The [ON] Key. This toggle key turns the computer on and off. After about 10 minutes of inactivity the computer automatically turns itself off to prolong battery life.* While the computer is off, Continuous Memory maintains the contents of main memory and the status of certain flags. To reset the computer (that is, to clear main memory and set all flags to default status):

1. Turn the computer off.
2. Hold down [←].
3. Press [ON].
4. Release [←].

The display will show **MEMORY LOST**.

The [USER] Key. This toggle key activates and deactivates the User keyboard, which is your redefined version of the Normal keyboard. The **USER** annunciator appears (and flag 27 is set) when the User keyboard is active.

The [PRGM] Key. This toggle key shifts the computer between Execution mode and Program mode. When you turn on the computer, it is in Execution mode—you can execute functions and programs. In Program mode you can write or edit programs; functions are stored as program steps to be executed later when you run the program in Execution mode. The **PRGM** annunciator indicates that the computer is in Program mode or that a program is running in Execution mode.

The [ALPHA] Key. This toggle key activates and deactivates the Alpha keyboard, which includes the blue letters on the lower face of the keys. The **ALPHA** annunciator appears (and flag 48 is set) when the Alpha keyboard is active. Pressing [ON] or [PRGM] deactivates the Alpha keyboard.

The Keyboards

This manual shows each function name in a color that indicates how to execute that function. The following overview of the keyboards covers this use of color and the basic purpose of each keyboard.

* Unless you execute [ON], which sets flag 44 (Continuous On). Flag 44 is cleared each time you turn on the computer.

The Normal Keyboard

The Normal keyboard comprises the functions printed in white on the upper face of the keys and the functions printed in gold above the keys. This is the default keyboard—it is active after Continuous Memory is cleared.

When you press ■ the **SHIFT** annunciator appears, indicating that a shifted function will be executed. The annunciator disappears when you press a second key (to execute the shifted function) or press ■ a second time (to cancel the shift command).

This manual represents an unshifted function by its name in black inside a black box, and a shifted function by its name in gold inside a gold box. For example, [LN] is the unshifted function on the top right key, and [e^x] is the shifted function. This rule applies to other keyboards too; for example, [AVIEW] is a shifted function on the Alpha keyboard.

When a key has a special meaning associated with the letter on the its lower face, that key is represented by the letter in black inside a black box. For example, the keystroke sequence that produces [RCL] Z would be [RCL] [·] [Z], with [Z] representing the [1] key.

The User Keyboard

The User keyboard is your customized version of the Normal keyboard. You can assign a function or global label to any key except the toggle keys or the ■ shift key. You can then execute that function, or start program execution at that global label, by pressing the redefined key on the User keyboard.

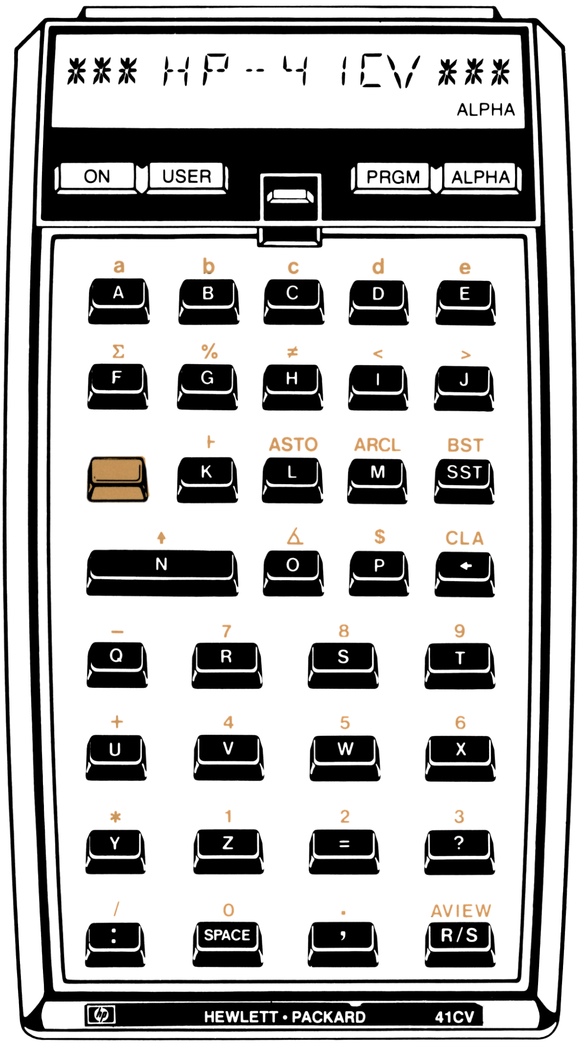
Because shifted key positions can be redefined as well, one key can execute four different functions, depending on whether the User keyboard is active and whether the ■ shift key is pressed first. The operation of the User keyboard is described in this section under "Redefining the User Keyboard," page 20.

Many functions are not on the Normal keyboard but can be assigned to the User keyboard. These are called nonkeyboard functions. This manual represents a nonkeyboard function by its name in blue inside a blue box.

The Alpha Keyboard

The Alpha keyboard comprises letters, functions, symbols, and digits considered as characters rather than numbers. The blue letters and symbols on the lower face of the keys are the unshifted characters on the Alpha keyboard. Digits 0 through 9 and the arithmetic symbols are shifted characters on the keys where they appear on the upper face. Shown on the next page is the entire Alpha keyboard, which includes functions and additional symbols in shifted positions.

The Alpha Keyboard



There are two distinct uses for the Alpha keyboard.

- To spell out a function or global label as a parameter for [ASN], [CLP], [COPY], [GTO], [LBL], or [XEQ]. In such cases the characters become part of the instruction.
- To key characters into the Alpha register. Here they are saved until you write over them or clear the register. The Alpha register is used to display your own messages, to specify file names and global labels for certain functions, and to manipulate bytes of data.

This manual shows unshifted characters on the Alpha keyboard in blue, shifted characters in gold. Note that a digit printed in gold represents an Alpha character while a digit printed in black represents a number on the Normal keyboard.

Keying In Numbers and Characters

Keying numbers into the X-register and keying characters into the Alpha register are similar processes. In both cases:

- When you enter the first digit or character, the display shows that digit or character followed by the *input cue* (_).
- The input cue indicates that the computer will append the next entry from the keyboard to the string of digits or characters in the display.
- When the input cue is displayed, you can correct your entry by pressing [←] to delete the rightmost digit or character.* The input cue then moves left to replace it.
- If the input cue is not displayed, entry has been terminated and the next entry from the keyboard will start a new number or Alpha string.*

Keying In Numbers

Up to 10 digits can be keyed into the X-register—additional digits will be ignored. The only keys used for digit entry are digit keys [0] through [9], [·], [CHS] (*change sign*), [EEX] (*enter exponent*), and [←]. Pressing any key other than a digit entry key, [■], or [USER] terminates digit entry—subsequent digits will be considered a new number.

Pressing [CLx] replaces the number in the X-register with zero; if you key in another number now, it will replace this zero. If there is only one digit in the display or if digit entry has been terminated, [←] has the same effect as [CLx].

---

* If you key 10 digits or a two-digit exponent into the X-register, the input cue will disappear because no additional digits are allowed. However, entry has not been terminated: your next entry *will not* start a new number, and pressing [←] *will* delete the rightmost digit.

Entering an Exponent. To enter a number in the form $a \times 10^b$, first key in the digits and decimal point for a and then press [CHS] if the number is negative. To enter more than eight digits for a, you must key in a decimal point somewhere to the left of the ninth digit.

Second, press [EEX]. Any digits to the right of the eighth digit will disappear but will remain internally. Enter one or two digits for the exponent b and press [CHS] if b is negative. If you press [EEX] without first entering a value for a, the computer sets a equal to 1.

Entering π. Pressing [π] has the same effect as keying in 3.141592654 and terminating digit entry.

Keying In Characters

In Execution Mode. If the Alpha keyboard is active and you are not specifying a parameter, the characters go into the Alpha register. For keyboard input to the Alpha register under program control, execute [AON] before the program pauses or halts for input, and then [AOFF] when execution resumes.

The Alpha register can hold up to 24 characters. As you key in the 24th character, a tone sounds to warn you that the Alpha register is full. If you key in a character when the Alpha register is full, the leftmost character is pushed out of the Alpha register and is lost.

Character entry is terminated by [ASTO], [BST], [SST], [AVIEW], [R/S], or by deactivating the Alpha keyboard. Character entry is restored by [⊢] (*append*) or by [ARCL].

Pressing [CLA] deletes all characters from the Alpha register. If character entry has been terminated, [←] has the same effect as [CLA].

In Program Mode. Up to 15 characters can be stored in a program line, which will be displayed with a leading T. The characters that follow are entered into the Alpha register when the program is run. To add a string of characters to the Alpha register without replacing the previous contents, begin the string with [⊢]. For example, you can load more than 15 characters into the Alpha register by using two program lines, beginning the second line with [⊢]. (The character ⊢ appears only when the program line is displayed; the "append function" is executed when the program is run.)

> **Note:** Alpha strings appear within quotation marks when listed by a printer or video monitor. Only program lines that begin and end with quotation marks are Alpha strings; if a listed program line is *not* within quotation marks, it is a function. Don't mistake an unfamiliar function name for an Alpha string—be sure to press [XEQ] before keying in the function name.

For an example of the use of [⊢] in a program, see lines 12 and 13 in the FINANCIAL CALCULATIONS program, 82 through 84 in the WORD GUESSING GAME program, or lines 174 and 175 in the BLACK-JACK program, all in section 9.

Status Annunciators

The status annunciators appear along the bottom of the display. In addition to the **USER**, **PRGM**, **ALPHA**, and **SHIFT** annunciators mentioned above, the following annunciators may appear.

- **BAT** indicates that the batteries are low. With alkaline batteries, about 5 to 15 days of operating time remain after **BAT** first appears. With the HP 82120A Rechargeable Battery/Reserve Power Pack, about 2 to 50 minutes of operating time remain. If you use the HP 82104A Card Reader or the HP 82153A Optical Wand, the operating time remaining will be reduced. For more information about batteries, refer to appendix B in the *HP-41CV Owner's Manual.*
- **GRAD** or **RAD** indicates that the computer is in Grads or Rads mode for trigonometric and rectangular/polar functions. If neither **GRAD** nor **RAD** appears, the computer is in Degrees mode.
- **0 1 2 3 4** indicates that the corresponding flag (00, 01, 02, 03, or 04) is set.

Numeric Display Format

The computer represents every number internally in the form $a \times 10^b$ where a is number with nine decimal places, $1 \leqslant |a| < 10$, and b is a two-digit integer, $0 \leqslant |b| < 100$. You can control how numbers are displayed without altering their internal representation. (If you do want to alter the number internally to match the display, refer to RND in section 2.) The format and punctuation you specify are maintained by Continuous Memory.

Formatting Numbers

There are three options for formatting numbers, which are selected by the functions FIX, SCI, and ENG.

FIX ***n*****.** This format displays numbers with up to n decimal places ($0 \leqslant n \leqslant 9$). If the integer portion of a number requires more than $(10 - n)$ digits, fewer than n decimal places will be displayed. For example, the default format is FIX 4, which displays numbers to four decimal places; but if a number has eight digits before the radix mark, only two decimal places will be displayed.

The last displayed digit is rounded up if the first hidden digit is 5 or greater. If the fractional portion of a number requires fewer than n digits, trailing zeros are added. If a number is too large or too small for the display, the format automatically and temporarily switches to SCI *n*.

SCI ***n*****.** This format displays numbers with one digit before and n digits after the radix mark ($0 \leqslant n \leqslant 9$), multiplied by a power of 10. For $n \leqslant 7$, the number is rounded to n decimal places. A maximum of 7 decimal places can be displayed, so SCI 8 or SCI 9 cause rounding to occur outside the display. (These formats can be useful when numbers are printed.)

ENG ***n*****.** This format displays a number with the same *digits* as SCI *n*, but with an exponent that is always a multiple of three. The radix mark is moved to the right to compensate for any change in the exponent.

Punctuation

Flags 28 and 29 control how periods and commas are used in number displays. In the U.S.A. a period is used as the radix mark (usually called the decimal point) to separate the integer and fractional parts of a number, and a comma is used as the separator mark between groups of digits in a large number. In some other countries, the comma is the radix mark and the period is the separator mark.

Flag 28 determines the roles of periods and commas. The default state for flag 28 is *set*, which produces the display normal for the U.S.A. Clearing flag 28 switches the roles of periods and commas to correspond with usage in some other countries.

Flag 29 determines whether a separator mark is displayed, regardless of which symbol represents the separator mark. The default state for flag 29 is *set*, which displays the separator mark. Clearing flag 29 suppresses all separator marks and, in the special case of FIX 0 format, suppresses display of the radix mark.

Standard Displays and Messages

The computer displays either the standard display or a message. The contents of the X-register are the standard display unless:

- The Alpha keyboard is active (and you're not keying in a parameter), in which case the contents of the Alpha register are the standard display.
- The computer is in Program mode, in which case the current program line is the standard display.
- A program is running, in which case the program execution indicator (⊁) is the standard display.

Any other display is a message such as a program's messages for the user (section 8). Examples covered in this section include the displays for parameter specification, function preview, the catalogs, and error messages. Flag 50 is set when the display contains a message.

Display Scrolling

To show more characters than the display can hold at one time, the computer "scrolls" the characters across the display until the last character enters the display. While the characters are moving you can press any key to bypass this process and immediately see the final display. The function whose key you pressed isn't executed.

Specifying Parameters

Certain functions require parameters to become complete commands. When the display shows the function name followed by one or more input cues (_), you must enter a parameter.

- For a numeric parameter such as a register address, flag number, local numeric label, program line number, and so on, observe how many input cues are shown and key in the desired digits. (You might need to add leading zeros, like 042 to specify program line 42.)
- For an Alpha parameter such as a function name or global label, press ALPHA to activate the Alpha keyboard, then spell out the name or label, and then press ALPHA again to complete parameter specification.

Indirect Parameter Specification

The parameters for most functions can be specified indirectly: rather than entering the parameter itself in response to the input cue, you enter the address of a register (the "indirect register") that contains the parameter. This feature is particularly useful when the value of the parameter depends on previous calculations in a program or when a routine is executed repeatedly to access sequential registers. In addition, the addresses for main memory registers $R_{(100)}$ through $R_{(318)}$ must be specified indirectly.

To specify a parameter indirectly:

1. Execute the function.
2. In response to the input cue, press . The display will show **IND _ _** after the function name.
3. Specify the indirect register.

The following examples demonstrate how indirect parameter specification works for three types of parameters. In each example R_{10} is the indirect register containing a parameter of 5; in the first example 5 is simply a number, in the second example 05 is an address, and in the third example 05 is a label.

Example. Suppose that R_{10} contains 5. If you execute TONE IND 10, the number in R_{10} becomes the parameter for TONE. Therefore, TONE IND 10 is equivalent to TONE 5 when R_{10} contains 5.

$$\left.\begin{array}{c} \boxed{\text{TONE}}\ \text{IND 10} \\ + \\ R_{10}\ \boxed{5} \end{array}\right\} \boxed{\text{TONE}}\ 5.$$

Example. Suppose that R_{10} contains 5. If you execute $\boxed{\text{STO}}$ IND 10, the address in R_{10} becomes the parameter for $\boxed{\text{STO}}$. Therefore, $\boxed{\text{STO}}$ IND 10 is equivalent to $\boxed{\text{STO}}$ 05 when R_{10} contains 5.

$$\left.\begin{array}{c}\boxed{\text{STO}}\ \text{IND 10}\\ + \\ R_{10}\ \boxed{5}\end{array}\right\}\ \boxed{\text{STO}}\ 05.$$

Indirect specification of an address—called *indirect addressing*—is the most common use for indirect parameter specification, and the most common use for indirect addressing is to access a series of registers by a looping routine in a program. For example, a loop containing $\boxed{\text{RCL}}$ IND 10, $\boxed{1/x}$, $\boxed{\text{STO}}$ IND 10 will replace the number in R_{05} with its reciprocal when R_{10} contains 5 (as illustrated above). The loop can then increment the address in R_{10} from 5 to 6 and start over, this time replacing the number in R_{06} with its reciprocal and incrementing the address in R_{10} from 6 to 7, and so on. (Loops are described in section 7, "Branching.")

Example. Suppose that R_{10} contains 5. If you execute $\boxed{\text{XEQ}}$ IND 10, the label in R_{10} becomes the parameter for $\boxed{\text{XEQ}}$. Therefore, $\boxed{\text{XEQ}}$ IND 10 is equivalent to $\boxed{\text{XEQ}}$ 05 when R_{10} contains 5.

$$\left.\begin{array}{c}\boxed{\text{XEQ}}\ \text{IND 10}\\ + \\ R_{10}\ \boxed{5}\end{array}\right\}\ \boxed{\text{XEQ}}\ 05.$$

You can also indirectly specify any global label listed in catalog 1 or any programmable function or global label listed in catalog 2, provided that the label doesn't exceed six characters.

Parameters can be indirectly specified for the following functions:

- Functions with register-address parameters.

  STO, RCL.
  STO +, STO −, STO ×, STO ÷.
  ASTO, ARCL.
  ISG, DSE.
  X<>, VIEW, ΣREG.
- XEQ, GTO.
- SF, CF, FS?, FC?, FS?C, FC?C.
- FIX, SCI, ENG.
- TONE.

Three programs in section 9 use indirect addressing. The CURVE FITTING program uses indirect addressing with XEQ and GTO (lines 27, 32, 60, 119 and 146) to specify which routine to use to handle the data, depending on which of the curve fits you are using.

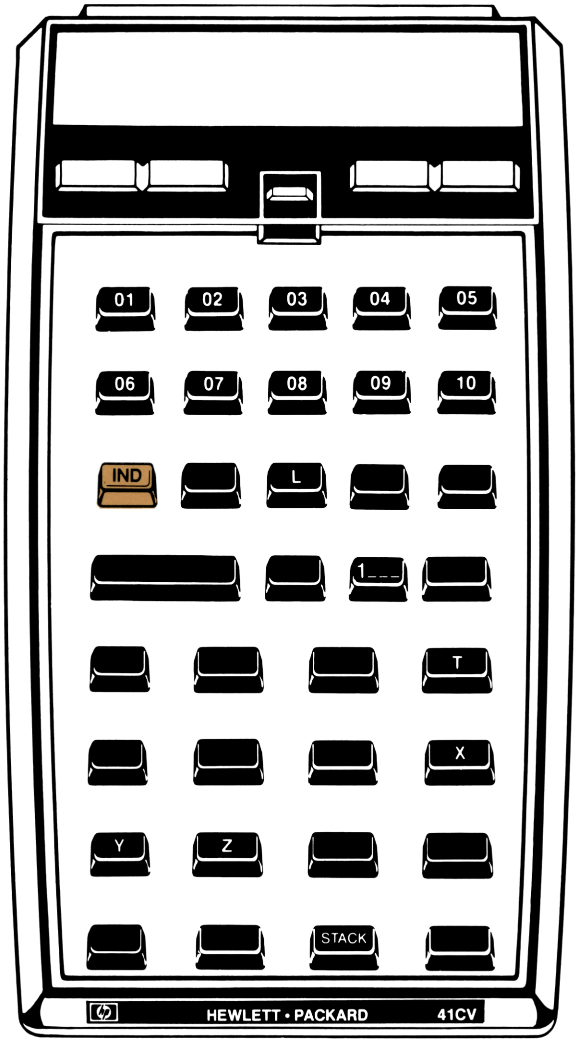
The WORD GUESSING GAME uses indirect addressing with ARCL and ASTO (lines 31, 40, 57 and 100) to store Alpha information in or recall Alpha information from sequential registers.

BLACKJACK uses indirect addressing with RCL, DSE, and STO (lines 19, 27 and 88).

Special Keys

The following diagram shows the keys that have special meanings when you're specifying a parameter for functions in catalog 3.

Special Keys for Specifying Parameters



Stack Register Addresses. To specify a stack register or the LAST X register, press [·] followed by [X], [Y], [Z], [T], or [L].

Program Line Numbers. To specify line numbers over 999, press [EEX]. The display will show 1 _ _ _. Then key in the remaining three digits.

Single-Key Parameter Specification. For convenience, you can specify a one-digit parameter of 0 through 9, or a two- or three-digit parameter of 1 through 10, by pressing the appropriate key in the two top rows. For example, when one, two, or three input cues are displayed, pressing [Σ+] enters a parameter of 1, 01, or 001. If only one input cue is displayed, pressing [TAN] enters a parameter of 0; if two or three input cues are displayed, pressing [TAN] enters a parameter of 10 or 010.

Redefining the User Keyboard

There is a nonprogrammable function that assigns functions and global labels to the User keyboard: [ASN] (*assign*).

To make an assignment:

1. Execute [ASN].
2. Press [ALPHA], key in the function name or global label, and press [ALPHA] again.
3. Press the key (or ■ and the key) to be redefined.

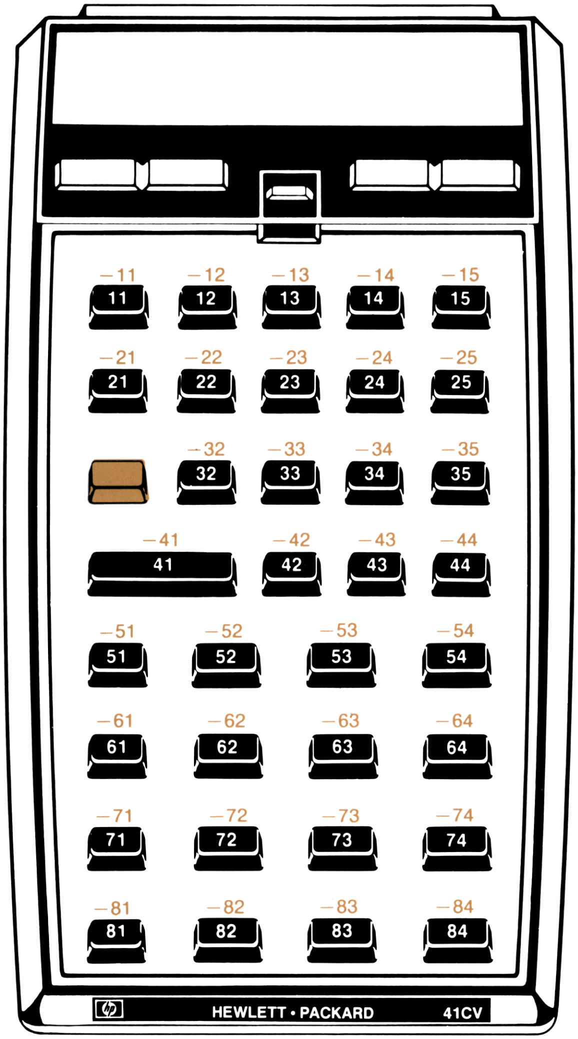
The following diagram shows the keycodes for the User keyboard. Note that:

- All keycodes have two digits.
- Keycodes for shifted locations are negative.
- You can't redefine the toggle keys or the shift key.
- You can redefine the [R/S] key. Your redefinition supersedes the "run" function in Execution mode and the [STOP] function in Program mode, but you can still press [R/S] to stop a running program.

When you assign a function listed in catalog 2 or 3, or a global label listed in catalog 2, the assignment is stored in User keyboard memory. (User keyboard memory is a part of main memory and is described in section 4.) However, when you assign a global label listed in catalog 1, that assignment is stored as a part of the label itself. If the label is deleted from program memory, the assignment is cancelled. If the program containing the assigned label is stored in extended memory, and if the User keyboard is active (flag 27 is set) when the program is recalled from extended memory, the assignment stored in the label will be reactivated.

The RPN PRIMER program, in section 9, redefines much of the keyboard. When you press one of the redefined keys, the assigned routine is executed instead of the normal function.

The User Keyboard



Restoring Normal Functions

To cancel the assignment to a redefined key:

1. Execute ASN.
2. Press ALPHA twice.
3. Press the appropriate key.

The Two Top Rows

There is a special type of program label, the local Alpha label, that is designed for use with the two top rows of the User keyboard. The name of each label corresponds to an Alpha character on the top two rows: A through E on the top row, F through J on the second row, and a through e on the shifted top row. Section 7, "Branching," discusses how to program with these labels; the discussion here covers only the conditions required to execute a local Alpha label on the User keyboard. These conditions are:

- The User keyboard is active.
- The current program contains the local Alpha label.
- You haven't redefined the key that corresponds to the local Alpha label.

These conditions combine with the general rules for the User keyboard to produce the following priorities. When you press a key on the top two rows of the User keyboard:

1. If you have assigned a function or global label to the key, that function is executed or program execution begins at that global label.
2. If you haven't redefined the key and the corresponding local Alpha label exists within the current program, execution begins at that local Alpha label.
3. If neither of the first two conditions is true, the Normal keyboard function—the one printed on (or above) the key—is executed.

Execution of a Normal keyboard function may take significantly more time when the User keyboard is active because the computer checks the higher priorities first. To avoid this delay when executing a Normal keyboard function, you can deactivate the User keyboard before pressing the key or else assign the Normal keyboard function to that key.

The FINANCIAL CALCULATIONS program in section 9 uses local Alpha labels A through E and a to store and calculate the various financial parameters.

Function Preview and Null

You can display the current meaning of a key, without necessarily executing the resulting function, by holding down the key. This preview is particularly helpful on the User keyboard when you're not sure which keys are redefined.

- If the function requires a parameter (one or more input cues appear), release the key. If you want to cancel the function, press [←].
- If the function doesn't require a parameter, you can either release the key to execute the function or else hold the key down until **NULL** is displayed to cancel the function.

In addition, there are four situations when a program line is previewed. (Assume that you release the key before **NULL** is displayed.)

- If the User keyboard is active and you press a key to which you've assigned a global label, that label is displayed and program execution begins at that label.
- If the User keyboard is active and you press a key that corresponds to a local Alpha label in the current program, **XEQ *label*** is displayed and program execution begins at that label.
- If you press [R/S], the current program line is displayed and program execution begins at the current program line.
- If you press [SST], the current program line is displayed and only the current program line is executed.

The Catalogs

There are three catalogs that enable you to review memory contents. The [CATALOG] function is not programmable. The rules of operation common to all catalogs are described first, followed by an overview of each catalog.

Basic Catalog Operation

Execute [CATALOG] *n* to start the listing of catalog *n*.

While the listing is running:

- Pressing any key except [R/S] and [ON] slows down the listing.
- Pressing [R/S] stops the listing.

While the listing is stopped:

- Pressing [SST] displays the next item in the catalog.
- Pressing [BST] displays the previous item in the catalog.
- Pressing [R/S] restarts the listing.
- Pressing [←] exits the catalog.

A printer in Trace mode will print a catalog listing.

Types of Catalogs

Catalog 1: User Programs. A list of all global labels and [END] instructions. With the permanent **.END.** (the final entry) appears the number of registers available for new programs.

You can use catalog 1 to make any program the current program: press [R/S] to stop the listing at that program's global label or [END] instruction, and then press [←] to exit the catalog. (Section 5.)

Catalog 2: External Functions. A list of all functions and programs currently available to the computer from peripheral devices and plug-in modules, plus all extended memory and time functions. A T precedes global labels for programs to distinguish them from functions.

Functions and programs are grouped by source. (Appendix D.)

Catalog 3: Standard Functions. An alphabetical listing of the standard functions of the HP-41. This listing shows the Alpha name for each function, which may differ from the name that appears on the keyboard. You need to know the Alpha name to assign a function to the User keyboard and to interpret program lines.

Error Messages

An operation that is illegal is never executed. If the attempted operation is a program instuction, the computer stops program execution and displays an error message.*

- To clear the error message from the display, press [←].
- To execute a different function, simply press the appropriate key—you don't need to clear the error message first.
- To discover which instruction caused the error, press [PRGM] to switch to Program mode. The display then shows the program line containing the illegal operation (or an XROM number if a missing plug-in module caused a **NONEXISTENT** error).

* Flags 24 and 25 can prevent certain anticipated errors from stopping program execution. These flags are described in section 6.

A list of error and status messages appears in appendix A. Many devices that plug into the computer have their own messages which may appear in the computer display. Refer to the literature for those devices to learn about such messages.

Section 2

The Automatic Memory Stack

Contents

Introduction

Numeric functions use four registers called the *automatic memory stack*. Numbers automatically move "up and down" in the stack when you enter numbers and perform calculations. The logic used is Reverse Polish Notation (RPN), which minimizes keystrokes and produces all intermediate results. If you are unfamiliar with RPN, refer to section 1 of the *HP-41CV Owner's Manual.* The RPN PRIMER program in section 9 will also help you use and understand RPN, because the program displays the contents of the stack during a calculation.

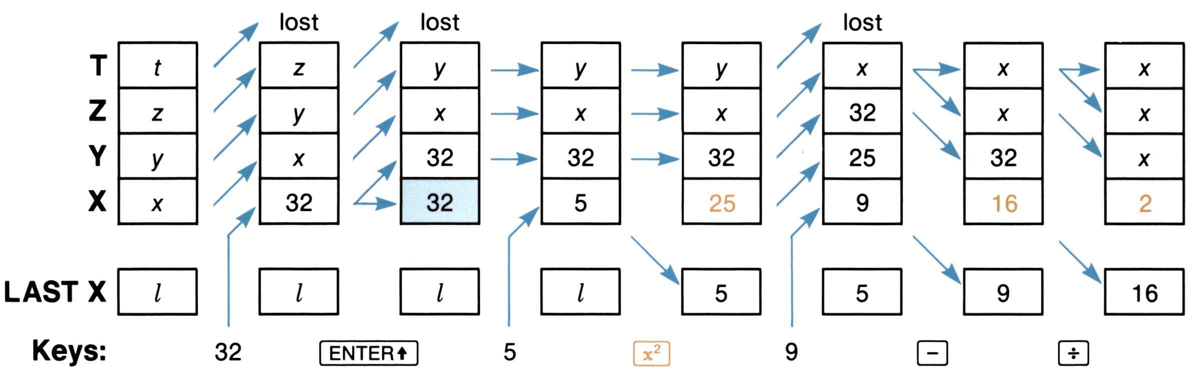
- The first topic in this section, "RPN Calculations," evaluates a typical numeric expression and describes the principles underlying use of the stack. Included is a method for constant arithmetic based on filling the stack with a constant.

- The second topic, "The LAST X Register," covers a special register closely related to the stack registers. The LAST X register is used for error correction and for a second method of constant arithmetic.
- The third topic describes other stack operations that give you more flexibility in using the stack, again emphasizing the repeated use of a constant.

RPN Calculations

The diagrams below show the contents of the automatic memory stack and the LAST X register following each step of an RPN calculation. Let *x*, *y*, *z*, *t*, and *l* represent numbers in the stack initially. The calculation evaluates the expression

$$\frac{32}{5^2 - 9}\ .$$



This example will be the basis for explaining how the stack works and how to use it efficiently.

Stack Lift and Stack Drop

The automatic movements of stack contents are called stack lift (moving upward in the diagram) and stack drop (moving downward).

Stack Lift. This usually occurs when a number is moved into the X-register. The numbers in the Y- and Z-registers are lifted into the Z- and T-registers; the number in the T-register is lost. In the example, stack lift occurs when 32 is keyed in, when ENTER↑ copies 32 into the Y-register, and when 9 is keyed in.

Stack Drop. This usually occurs when a function combines the numbers in the X- and Y-registers. The number in the Z- and T-registers are dropped into the Y- and Z-register; the number in the LAST X register is lost. In the example, stack drop occurs when − and ÷ are executed.

Using ENTER↑

Pressing ENTER↑ separates two numbers keyed in one after the other (32 and 5 in the example). This copies the number in the X-register (32) into the Y-register. The copy left in the X-register is replaced by the next number keyed in (5) because ENTER↑ *disables* stack lift.

Enabling/Disabling Stack Lift

Nearly all functions enable stack lift: the stack will lift if you place a number in the X-register *after* executing the stack-lift enabling function. However, four functions disable stack lift and others are neutral.

Stack-Lift Disabling Functions. The four functions that disable stack lift are ENTER↑, CLx, Σ+, and Σ−. If you execute one of these functions and then place a number in the X-register, that number will *replace* the previous contents and the Y-, Z- and T-registers will not be affected. Stack diagrams show when stack lift is disabled by shading the X-register, indicating that its contents will be replaced.

Neutral Functions. The following functions neither enable nor disable stack lift, but maintain the previous status:

- The toggle keys (ON, USER, PRGM, ALPHA).
- The backarrow key (←) during digit or character entry.
- The shift key (■).
- Catalogs 1, 2, and 3.

Order of Entry

Two major considerations affect the order in which you should enter operands. You can save many keystrokes by observing the following rules, although sometimes you must choose between them.

Nested Terms. For expressions with terms nested in parentheses, calculate the innermost term first and then use that result in the simplified expression. If two nested terms must be calculated before you can combine them, the automatic memory stack saves the result of the first term while you evaluate the second term. The example in "Polynomial Expressions" below demonstrates this rule.

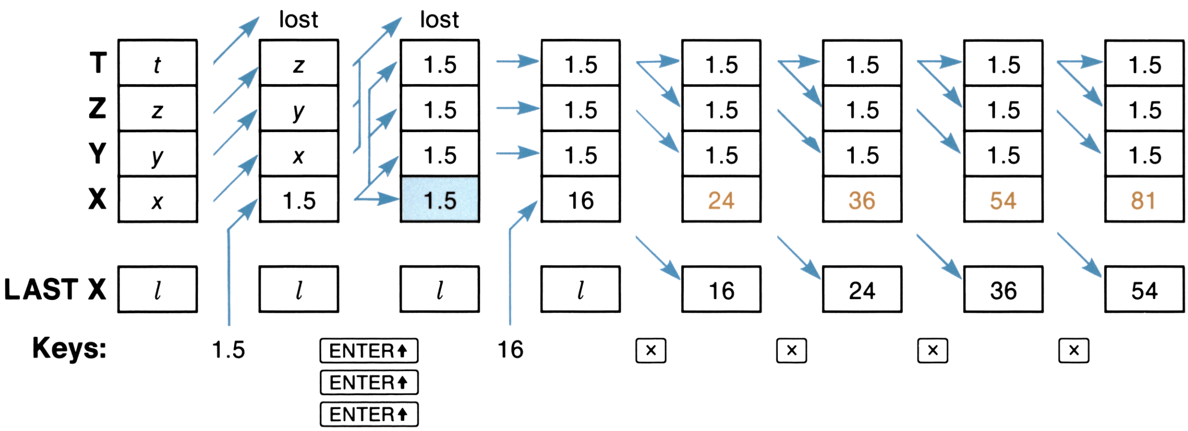
Noncommutative Functions. Functions like subtraction and division are called *noncommutative* because the order of the operands is essential: $5 - 3 \neq 3 - 5$, and $5 \div 3 \neq 3 \div 5$. For expressions involving noncommutative functions, enter or calculate the number that must be in the Y-register before entering or calculating the number that must be in the X-register. The previous example demonstrates this rule twice.

- The numerator (32) is entered before the denominator ($5^2 - 9$) is calculated.
- The term 5^2 is calculated before 9 is subtracted from it.

Filling the Stack

Note in the last three steps of the previous example how x propagates from the T-register into the Y- and Z-registers. This consequence of stack drop can keep the Y-register filled with a constant, as demonstrated in the next two examples. This technique is particularly appropriate when the constant must be in the Y-register for noncommutative operations like [−] and [÷]. (In contrast, [LASTx] supplies the constant in the X-register.)

Cumulative Growth. Suppose that you want to calculate the growth of a quantity that starts at a value of 16 and increases by 50% each period. First fill the stack with the growth factor (1.5) and key the starting value (16) into the X-register. Then press [×] to calculate the value after the first period and press [×] again for each subsequent period.



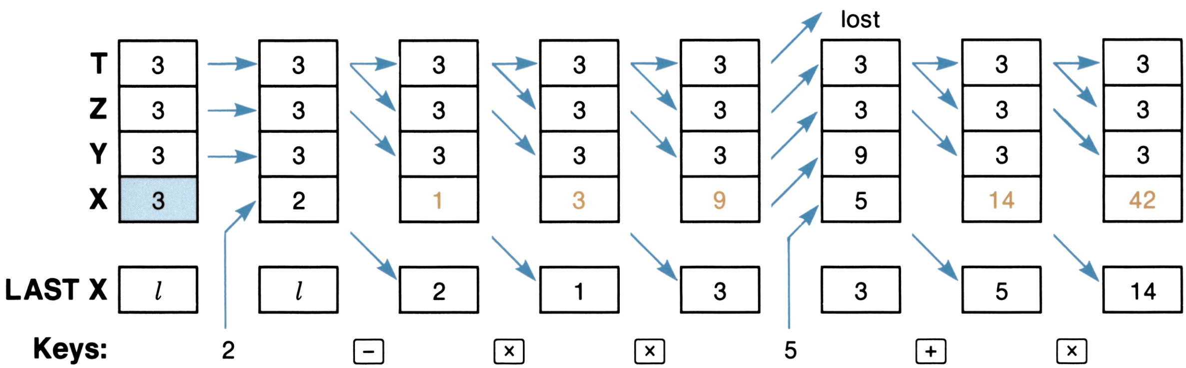
Polynomial Expressions. Filling the stack aids the evaluation of a polynomial, which requires several copies of the variable. For efficiency, use Horner's Method to rewrite the polynomial in a nested fashion that eliminates exponents greater than 1. Suppose that you want to evaluate

$$x^4 - 2x^3 + 5x$$

for $x = 3$. First, rewrite the polynomial to eliminate the exponents.

$$\begin{aligned} x^4 - 2x^3 + 5x &= (x^3 - 2x^2 + 5)x \\ &= ((x^2 - 2x)x + 5)x \\ &= (((x - 2)x)x + 5)x \end{aligned}$$

Then fill the stack with the variable by pressing 3, ENTER↑, ENTER↑, ENTER↑, and execute the steps below. Note that the calculation begins at the innermost nested term.



Once you are familiar with Horner's Method you can key in the steps for a polynomial without actually rewriting it. For example, the steps to evaluate the polynomial

$$ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

after filling the stack with the variable are:

a, ×, b, +, ×, c, +, ×, d, +, ×, e, +, ×, f, +.

- Note that coefficients (except the first and last) are followed by + and ×. (There is no previous result to add to first coefficient, and the last coefficient isn't multiplied by any power of the variable.)
- If the first coefficient is 1, start with the second coefficient. (The variable is already in the X-register.)
- For negative coefficients you may enter a positive value and substitute − for + following that coefficient.
- When there is no term for a power of x, just press ×. (In effect, this enters a coefficient of 0 for that power.)

Noncumulative Results. You can also use constant arithmetic to perform a series of unrelated (noncumulative) operations with a constant. After each calculation, press ← to clear the X-register before you key in the next operand. This disables stack lift, preventing the previous result from displacing the constant in the Y-register.

The LAST X Register

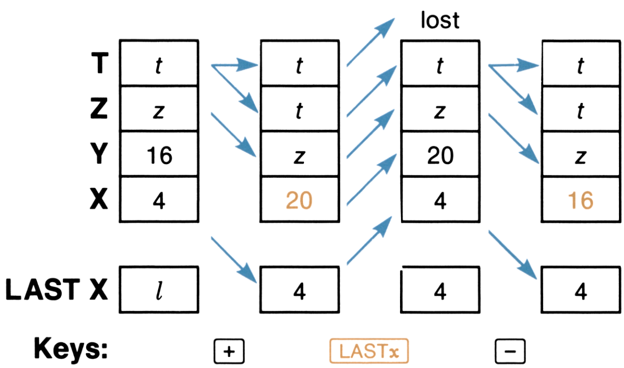
The LAST X register holds the *x*-operand from the last numeric function (except CHS). To recall this number to the X-register, press LASTx. This enables you to recover from errors and to retrieve an operand for further calculations.

Correcting Errors

One-Number Function Errors. If you execute the wrong one-number function, you can recover from your error as follows:

1. Press ←. This replaces the incorrect result with zero and disables stack lift.
2. Press LASTx. This recalls your operand, which replaces the zero in the X-register.
3. Continue your calculation with the correct function.

Two-Number Function Errors. If you make a mistake with a function like + or ÷, you can use LASTx and the inverse function (− or ×) to recover. Suppose that you made a mistake in adding two numbers. Press LASTx and then − as shown below. The nature of your mistake determines how you should continue; the alternatives are listed after the diagram.

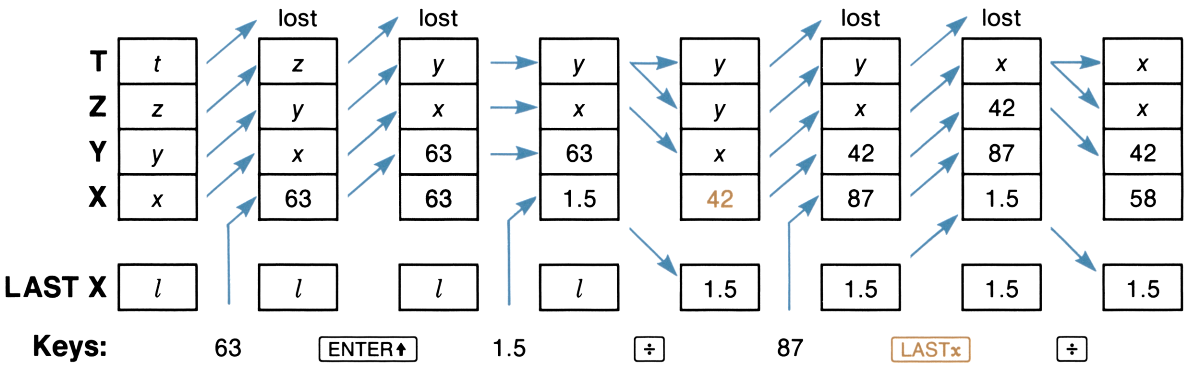


- If you wanted to multiply instead of add, execute LASTx again to return the stack to its original state, and then multiply.
- If 16 was the wrong number to add, press ← to clear 16, key in the correct number, execute LASTx to recover 4, and then add.
- If 4 was the wrong number to add, key in the correct number and then add.

Errors with some other types of two-number functions are even easier to correct. For example, you can cancel the effect of P→R by executing R→P, and you can correct errors with % and %CH as you would for a one-number function. To correct errors with other functions, determine how the function affects the stack, and then reverse that process.

Constant Arithmetic

The following example shows how to retrieve a constant for further calculations. Suppose that you want to divide both 63 and 87 by a factor of 1.5. This constant factor is entered second (after 63) to be in the X-register for the first calculation, and is subsequently maintained in the LAST X register.



This technique is particularly appropriate when the constant must be in the X-register for noncommutative operations like [−] and [÷]. (In contrast, constant arithmetic using stack drop supplies the constant in the Y-register.)

Other Stack Operations

You can consider the four stack registers as two pairs of registers. The X- and Y-registers are the center of almost all activity, while the Z- and T-registers are like storage registers connected by stack lift and stack drop to the more active X- and Y-registers. If you make an extra copy of a number while it's in the X-register or retrieve a copy from the LAST X register, you can temporarily store that copy in the higher stack registers and retrieve it later.

To take full advantage of the Z- and T-registers, plan ahead when you're programming a series of calculations. Figure out where the operands must be for each step, work backwards from the final calculation, and use the operations in the remainder of this section to link the result of one calculation with the input for the next. This efficient use of the stack saves program memory and reduces the need for storage registers.

Exchanging Stack Contents

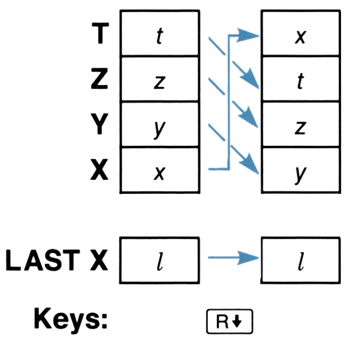
Exchanging the X- and Y-registers. Executing [x≷y] (*X exchange Y*) exchanges the contents of the X- and Y-registers. This function has several uses:

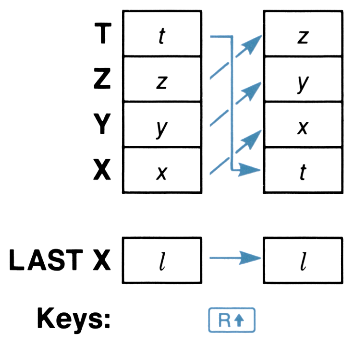
- To examine the contents of the Y-register. Press [x≷y], examine the display, and then press [x≷y] again to restore the numbers to their original order. This is useful when a function returns results to both the X- and Y-registers, as do the statistics functions and polar/rectangular coordinate conversions.
- To switch numbers that are in the wrong order for noncommutative operations such as subtraction and division.
- To rearrange the contents of the stack in combination with [R↓] or [R↑]; refer to "Rolling the Stack" below.

Exchanging X and Other Stack Registers. To exchange the contents of the X-register with a stack register or the LAST X register, execute [X<>] and then press [·] followed by [Y], [Z], [T], or [L]. Refer to "Stack Register Arithmetic" below for an example of this function's use.

Rolling the Stack

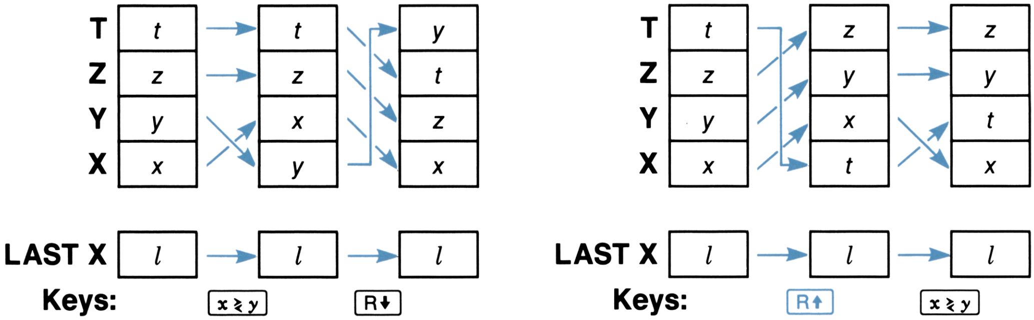
The [R↓] (*roll down*) and [R↑] (*roll up*) functions shift all stack contents without duplicating or losing any data.





Note that the LAST X register is unchanged. To review all numbers in the stack, press either [R↓] or [R↑] four times. Each number is displayed when it is rolled into the X-register, and the stack returns to its original state after four shifts.

Use [R↓] and [R↑] in combination with [x⋛y] to exchange stack registers other than the X-register. You can rearrange the stack in any order with these functions; here are two simple examples.



Store and Recall

You can duplicate any number in the stack by executing [STO] or [RCL] and then specifying a stack register. Both functions result in the X-register and the specified register containing the same number.

Store. To copy the number in the X-register into a stack register or the LAST X register, press [STO] [·] followed by [Y], [Z], [T] or [L]. The number in the specified register is lost.

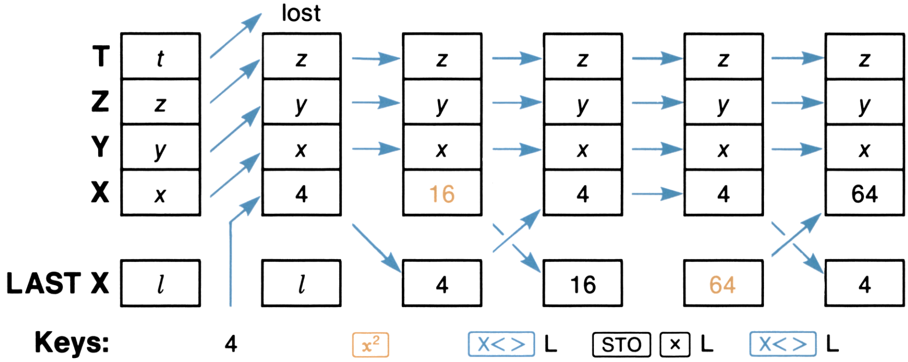
Recall. To copy the number in a stack register or the LAST X register into the X-register, press [RCL] [·] followed by [Y], [Z], [T], or [L]. The number in the T-register is lost as the stack lifts (unless stack lift is disabled).

Register Arithmetic

You can combine the number in the X-register with any stack register by pressing [STO] [+] [·], [STO] [−] [·], [STO] [×] [·], or [STO] [÷] [·] followed by [X], [Y], [Z], [T], or [L]. Remember that the order of the operands is essential for subtraction and division; the operand in the specified register corresponds to the operand in the Y-register for stack arithmetic. Register arithmetic in the stack differs in several ways from normal arithmetic in the stack:

- The result is placed in the specified register.
- The X-register is unchanged (unless you specify it as the parameter).
- The LAST X register is unchanged (unless you specify it as the parameter).
- The stack doesn't drop.

The following routine cubes the number in the X-register and places the original value in the LAST X register without disturbing the other stack registers.



Clearing the Stack

To place zeros in the X-, Y-, Z-, and T-registers, execute CLST (*clear stack*). The LAST X register is unchanged.

Section 3

Numeric Functions

Contents

Introduction

This section describes the numeric functions in the computer. All one- and two-number functions operate in the stack; their actions are shown by stack diagrams. Although data for the statistical functions are entered from the stack, they are accumulated in statistics registers in main memory. The results of operations on these accumulations are then returned to the stack. Certain other functions that involve calculations but do not necessarily return results to the stack (such as register arithmetic and [ISG]) are not included here.

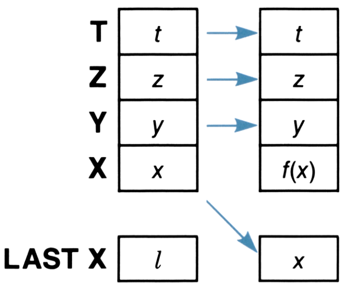
There are three error conditions that can result from numeric functions.

1. If you try to calculate with an operand that is illegal for that function (such as division when $x = 0$), a **DATA ERROR** results.
2. If you try to calculate with an operand that is not a number, an **ALPHA DATA** error results. Note that a string of Alpha digits from the Alpha register is not a number.
3. If you attempt a calculation that would produce a number with magnitude greater than $9.999999999 \times 10^{99}$, an **OUT OF RANGE** error results. (Statistical accumulations [Σ+] and [Σ−] are exceptions.)

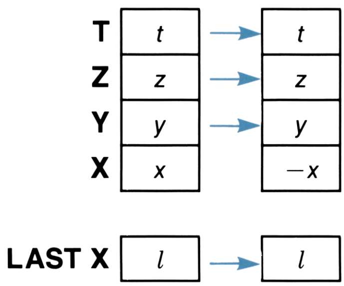
The computer does not execute a function that causes an error condition. Unless flag 25 is set, a **DATA ERROR** or **ALPHA DATA** error will stop program execution (if a program is running) and display the error message; an **OUT OF RANGE** error will stop execution and display the error message unless either flag 24 or 25 is set.

One-Number Functions

One-number functions replace the operand in the X-register with the result, save the operand in the LAST X register, and leave the Y-, Z-, and T-registers unchanged. $f(x)$ represents the result in the stack diagram on the left. The only exception is [CHS] (*change sign*), shown on the right, which doesn't save the operand.



One-Number Functions



[CHS]

General Functions

Reciprocal. Executing [1/x] returns the reciprocal of x.

Square and Square Root. Executing [x^2] returns the square of x. Executing [$\sqrt{x}$] returns the positive square root of x.

Factorial. For a positive integer n, executing [FACT] returns $n! = n\,(n-1)\,(n-2)\ldots 1$.

Number-Alteration Functions

Absolute Value and Sign. Executing ABS returns $|x|$, the absolute value of x. Executing SIGN returns:

1 if $x \geqslant 0$,

-1 if $x < 0$,

0 if the X-register contains Alpha data.

Integer Part and Fractional Part. These functions reduce a number to its integer part or its fractional part. For example, if the X-register contains 777.888, executing INT returns 777 or executing FRC returns 0.888.

Round. Recall that the display-format functions affect only how a number is displayed, not its internal representation. To round the internal representation of the number in the X-register:

1. Set the display format to the number of decimal places that you want the rounded number to contain.
2. Execute RND.

For example, to round a number to the nearest integer, execute FIX 0 and then RND.

Trigonometric Operations

Angular Modes. The angular mode determines how the computer interprets numbers as angles. Your choice of angular mode is maintained by Continuous Memory. These functions alter only the angular mode; they do not alter any numbers currently in the computer.

- Execute RAD to select *Radians* mode. The **RAD** annunciator appears, indicating that numbers will be interpreted as angles expressed in radians. (There are 2π radians in a circle.)
- Execute GRAD to select *Grads* mode. The **GRAD** annunciator appears, indicating that numbers will be interpreted as angles expressed in grads. (There are 400 grads in a circle.)
- Execute DEG to select *decimal Degrees* mode. This is the default angular mode; when neither the **RAD** nor the **GRAD** annunciator appear, numbers will be interpreted as angles expressed in degrees. Digits following the decimal point in the argument are interpreted as a decimal fraction of one degree, not as minutes and seconds.

Trigonometric Functions.

- SIN (*sine*) and SIN^{-1} (*arc sine*).
- COS (*cosine*) and COS^{-1} (*arc cosine*).
- TAN (*tangent*) and TAN^{-1} (*arc tangent*).

Conversions

Degrees/Radians Conversions. Execute D–R (*degrees to radians*) to convert a number expressing an angle in decimal degrees into the number that expresses the same angle in radians. For the inverse conversion, execute R–D (*radians to degrees*).

Hours-Minutes-Seconds/Decimal Hours Conversions. Hours and degrees can be expressed in HMS (*hours-minutes-seconds*) format rather than the normal decimal format. The first two digits following the decimal point are interpreted as minutes, the next two digits as seconds, and any subsequent digits as a *decimal* fraction of seconds. For example,

$$\begin{aligned} HH.MMSSssss &= HH \text{ hours} + MM \text{ minutes} + SS.ssss \text{ seconds (HMS format)} \\ &= HH + MM/60 + SS.ss/3600 \text{ (decimal format)} \end{aligned}$$

To convert a number in decimal format into HMS format, execute HMS (*to hours-minutes-seconds*). For the inverse conversion, execute HR (*to decimal hours*).

Decimal/Octal Conversions. To convert a decimal integer into its octal (base 8) equivalent, execute OCT (*to octal*). To convert an octal integer into its decimal (base 10) equivalent, execute DEC (*to decimal*).

Logarithmic and Exponential Functions

Common Logarithmic and Exponential Functions. Press LOG to calculate the common logarithm (logarithm to base 10) of the number in the X-register. Press 10^x to calculate 10 raised to the power of the number in the X-register.

Natural Logarithmic and Exponential Functions. Press LN to calculate the natural logarithm (logarithm to base e) of the number in the X-register. Press e^x to calculate e raised to the power of the number in the X-register.

Hyperbolic functions, inverse hyperbolic functions, and certain financial calculations evaluate the expressions $\ln(1 + x)$ and $e^x - 1$ for arguments near zero and with results also near zero. To allow greater accuracy in such calculations, LN1+X and E↑X–1 evaluate these expressions directly.

- LN1+X computes $\ln(1 + x)$.
- E↑X–1 computes $e^x - 1$.

Two-Number Functions

All two-number functions use operands in the X- and Y-registers; most return a single number to the X-register and cause the stack to drop. (Percentages and polar/rectangular coordinate conversions are exceptions.)

Basic Arithmetic

Stack diagrams for [+], [−], [×], and [÷] appear in the previous section. Remember the order of entry for subtraction and division: for x in the X-register and y in the Y-register,

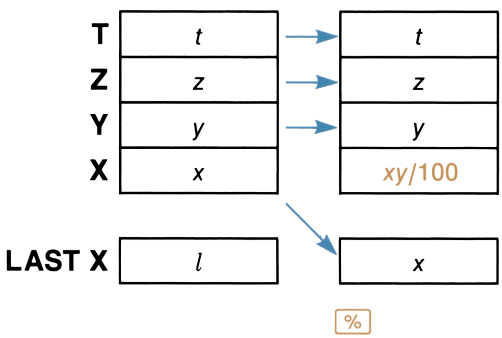
- Subtraction returns $y - x$ (not $x - y$).
- Division returns y/x (not x/y).

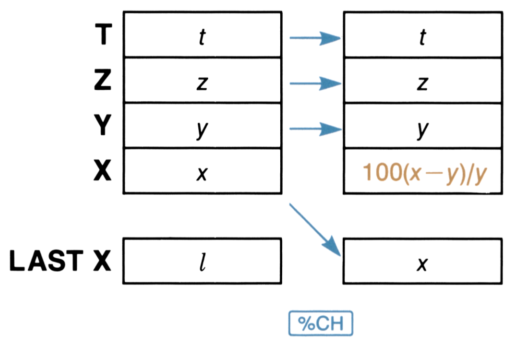
Time Arithmetic

To add or subtract numbers that are in HMS (hours-minutes-seconds) format, use [HMS+] (*hours-minutes-seconds add*) or [HMS−] (*hours-minutes-seconds subtract*). The order of entry and stack drop are identical to those for normal addition and subtraction.

Percentages

The two percentage functions use the number in the Y-register as a base and alter the number in the X-register, expressing it in terms of the base. Note that the base number in the Y-register is unaltered and that the stack doesn't drop.





Percent. To calculate a percentage, place the base number in the Y-register and the percent rate in the X-register, and then execute [%].

Percent Change. To calculate the increase or decrease from one number to another, place the first (base) number in the Y-register and the second number in the X-register, and then execute [%CH]. The increase or decrease is returned as a positive or negative percentage of the first (base) number.

Percent of Total. To calculate the percentage that one number is of another number:

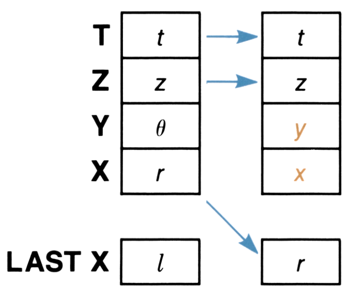
1. Place the total (base) number in the Y-register and the number to be converted to a percentage in the X-register.
2. Execute [1/x].
3. Execute [%].
4. Execute [1/x].

Polar/Rectangular Conversions

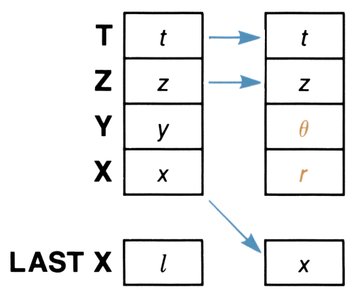
A point in a plane can be described by either polar or rectangular coordinates. Polar coordinates are r (magnitude) and θ (angle); rectangular coordinates are x (horizontal) and y (vertical). (An illustration of these coordinates is on page 53 in the *HP-41CV Owner's Manual.*) Two functions, [P→R] and [R→P], convert between polar and rectangular coordinates.

- To convert polar coordinates to rectangular coordinates, execute [P→R] (*polar to rectangular*).
- To convert rectangular coordinates to polar coordinates, execute [R→P] (*rectangular to polar*). The resulting θ will have the same sign as the y-coordinate input.

As input or output, θ is interpreted according to the current angular mode. In the stack diagrams below, note the order of the coordinates in the stack and that the stack doesn't drop. Press [x≷y] to see the result returned to the Y-register.



[P→R]



[R→P]

Other Two-Number Functions

Raising a Number to a Power. To raise a number to a power, place the base number in the Y-register and the power in the X-register, and then execute [y^x]. Stack drop is the same as for arithmetic functions. Legal values for x depend on the value of y:

- If y is positive, x can be any number.
- If y is negative, x must be an integer.

- If y is zero, x must be positive.

Any other combination causes a **DATA ERROR**.

Finding Roots. To calculate the nth root of a number:

1. Place the number in the Y-register.
2. Place n in the X-register.
3. Execute [1/x].
4. Execute [y^x].

Modulo. For positive integers x in the X-register and y in the Y-register, executing [MOD] calculates the remainder when y is divided by x ("y mod x"). For example, you can test whether y is evenly divisible by x by executing [MOD] and testing whether the result is zero. Stack drop is the same as for arithmetic functions.

You can also use [MOD] with numbers that are not positive integers. The general equation for y mod x is $y - x <y/x>$, where $<y/x>$ represents the largest integer not larger than y/x. Performing y mod x when $x = 0$ returns an answer of y.

Statistics

There are two stages in performing statistical calculations. First you enter data from the stack; the computer accumulates intermediate statistics from this data. Then you execute statistical calculations; the computer uses the intermediate statistics to calculate the overall results, which are returned to the stack. Basic statistical operations are described in section 5 of the *HP-41CV Owner's Manual.*

For an example of using statistical functions in a program, refer to the CURVE FITTING program in section 9. The program uses [ΣREG] to move the statistical registers, and [Σ+] and [Σ−] to sum and correct the data.

Statistics Registers

The statistics registers are a block of six data registers in main memory that hold the intermediate statistics accumulated from your data. When the computer memory is reset, the statistics registers are R_{11} through R_{16}.

- You can assign other storage registers to be the statistics registers by executing [ΣREG] and specifying the address of the first register in the block you select. This assignment is maintained by Continuous Memory.
- To place zeros in all six statistics registers, execute [CLΣ].

The statistics registers accumulate the following intermediate statistics from your data in the X- and Y-registers.

The Statistics Registers

Register	Contents	
R_{11}	Σx	Summation of x-values.
R_{12}	Σx^2	Summation of squares of x-values.
R_{13}	Σy	Summation of y-values.
R_{14}	Σy^2	Summation of squares of y-values.
R_{15}	Σxy	Summation of products of x- and y-values.
R_{16}	n	Number of data points accumulated. (Displayed.)

Entering Data

Accumulating Data Points. When you press [Σ+]:

- The results of calculations using the numbers in the X- and Y-registers are added to the first five statistics registers. If this causes the contents of a register to exceed $\pm 9.999999999 \times 10^{99}$, there is no overflow error; the overflowed register contains $\pm 9.999999999 \times 10^{99}$.
- The number of data points n in the sixth register is incremented and its current value is returned to the X-register.
- The number previously in the X-register is saved in the LAST X register.
- Stack lift is disabled, so the next data entered will replace n in the X-register.

You can accumulate either one-value or two-value data points, as discussed in part I. If you are accumulating only x-values, clear the Y-register first (0 [ENTER↑]). Because [Σ+] and [Σ−] disable stack lift, the Y-register will remain clear while you accumulate x-values.

Error Correction. To correct erroneous data that have been accumulated:

1. Re-enter the erroneous data. If you just accumulated the erroneous data, simply press [LASTx] to retrieve them. (The erroneous y-value is still in the Y-register and the erroneous x-value was saved in the LAST X register.)
2. Press [Σ−]. This function acts similarly to [Σ+] except that the results are subtracted from (rather than added to) the first five statistics registers, and the sixth register is decremented (rather than incremented).
3. Enter the correct data.
4. Press [Σ+].

Limitation on Data Values. The computer might be unable to perform some statistical calculations if your data values differ by a relatively small amount. To avoid this, you should normalize your data by entering the values as the difference from one value (such as the mean). This difference must then be added back to any calculations of the mean. For instance, if your x-values were 665999, 666000, and 666001, you should enter the data as -1, 0, and 1; then add 666000 back to the relevant results.

Mean

Executing MEAN returns the arithmetic average $\bar{x}$ of the accumulated x-values to the X-register and the arithmetic average $\bar{y}$ of the accumulated y-values to the Y-register, according to the following formulas:

$$\bar{x} = \frac{\Sigma x}{n}, \qquad \bar{y} = \frac{\Sigma y}{n}.$$

Press $x \gtrless y$ to display the resulting y-value. The number previously in the X-register is saved in the LAST X register; the number previously in the Y-register is lost.

Standard Deviation

Executing SDEV returns the sample standard deviation s_x of the accumulated x-values to the X-register and the sample standard deviation s_y of the accumulated y-values to the Y-register, according to the following formulas:

$$s_x = \sqrt{\frac{n\Sigma(x^2) - (\Sigma x)^2}{n(n-1)}}, \qquad s_y = \sqrt{\frac{n\Sigma(y^2) - (\Sigma y)^2}{n(n-1)}}.$$

Press $x \gtrless y$ to display the resulting y-value. The number previously in the X-register is saved in the LAST X register; the number previously in the Y-register is lost.

Section 4

Main Memory

Contents

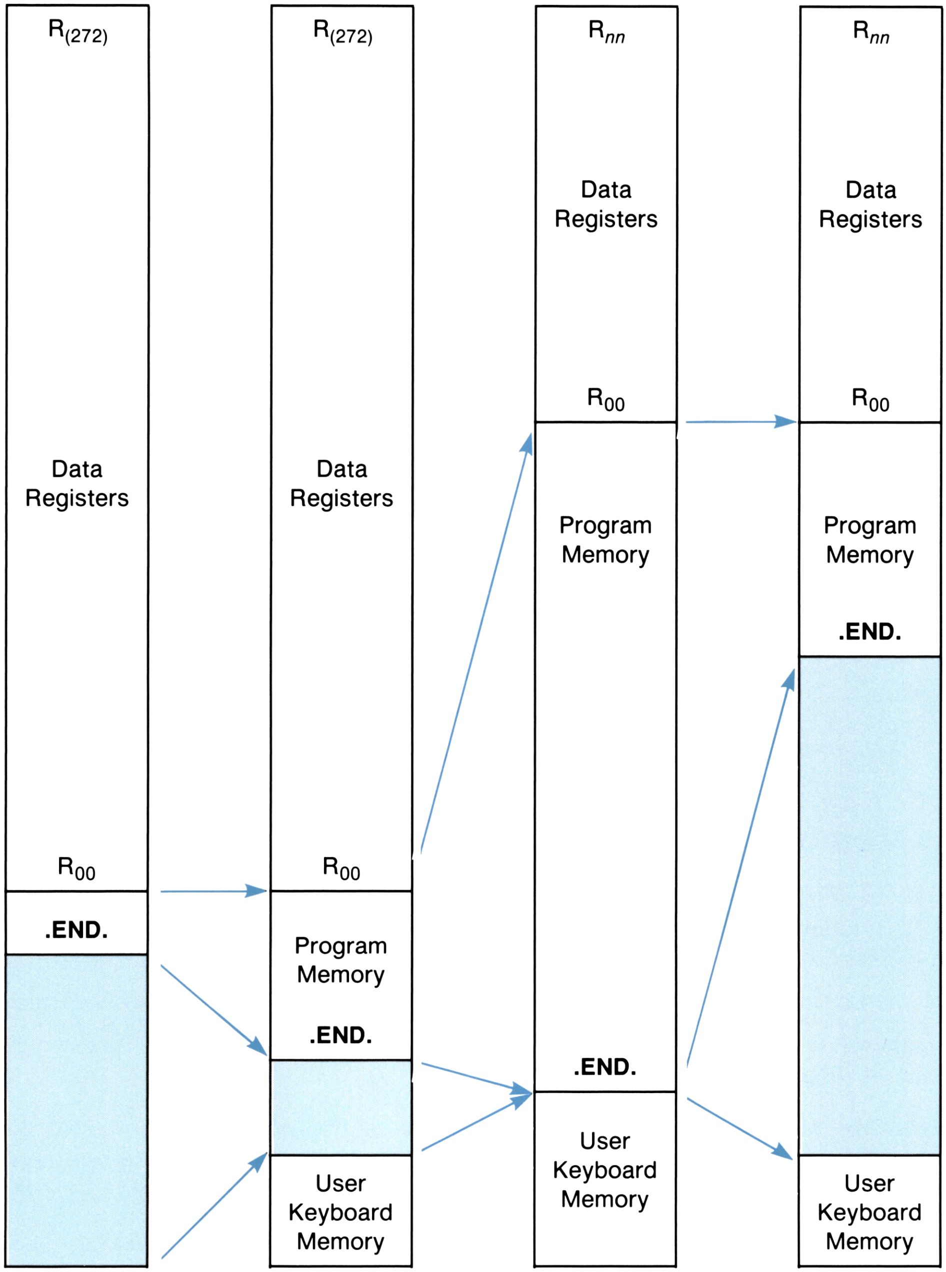
Organization

Main memory contains 319 registers divided into two major groups.*

- One group contains the data storage registers. The number of main memory registers allocated to data storage changes only when you execute a function to specify the allocation.
- The other group contains programs, key redefinitions, and uncommitted registers. The uncommitted registers are automatically committed to programs, and key redefinitions as needed. However, the size of this group as a whole changes only when you change the number of registers allocated to data storage.

* Main memory actually contains 320 registers, but program memory always contains at least one register for the permanent **.END.**

Main Memory Configurations



The preceding diagram illustrates four configurations of main memory—each column represents all of main memory at one time. The leftmost column shows the default configuration, 273 registers allocated to data storage and 46 registers for all other purposes. The columns to the right show main memory at three later times. The computer handles most of these details automatically, but understanding main memory will help you use it more effectively.

The first column represents the default configuration after Continuous Memory is cleared. There are 273 registers for data storage with the largest-numbered register at the top. The first register below the data register block holds the permanent **.END.**, which marks the bottom of program memory. The uncommitted registers below the permanent **.END.** are available for programs and key redefinitions.

The second column shows main memory after you've entered programs and assigned functions to keys. Program memory consumes uncommitted registers as the permanent **.END.** is pushed down by new program lines. User-keyboard memory consumes one uncommitted register for every two assignments.

The third column shows the result after you allocate fewer registers to data storage, but write more programs, and redefine more keys. When all registers are committed (as in this column), any operation that would consume main memory registers causes the computer to display **PACKING** and then **TRY AGAIN**.

If packing doesn't produce a sufficient number of uncommitted registers, you'll have to reduce the size of the data storage block or delete other memory contents. You can review the contents of program memory by executing [CATALOG] 1.

The fourth column shows the reappearance of uncommitted registers after you delete programs and User-key assignments. You could gain even more uncommitted registers by allocating fewer registers to data storage.

Program Memory

When Continuous Memory is cleared, program memory contains only the permanent **.END.**. If you press [GTO] [.] [.] and key in a program, each instruction is added just before the permanent **.END.** which moves down to make room. As a result:

- The first instruction of the program you keyed in first is at the top of program memory.
- The last instruction of the program you keyed in most recently precedes the permanent **.END.** at the bottom of program memory.

Catalog 1 shows the number of uncommitted registers along with the permanent **.END.** (**.END. REG** ***nnn***). There can be up to six bytes (nearly a full register) available in addition to *nnn* registers.

Program Lines

Each function, number, or Alpha string in a program is considered to be a separate program line. The number of *program lines* depends on how many functions, numbers, and Alpha strings are in the program; the number of *registers and bytes* occupied by these program lines depends on the particular functions and the lengths of the numbers and Alpha strings:

- Functions require from one to four bytes, depending on the particular function (and on the parameter if one is needed). The number of bytes required for each function is listed in the Function Tables at the back of this manual.
- Functions with global labels as parameters require one byte per character in addition to their normal length.
- Numbers require one byte per digit, plus another byte for each [.], [CHS], or [EEX] keyed in with the number.
- Alpha strings require one byte per character, plus one additional byte for the entire string.

Null Bytes

Usually the first byte of an instruction immediately follows the last byte of the previous instruction, but sometimes there are null bytes between instructions. Null bytes result from:

Deleting an Instruction. When you delete an instruction, the bytes it occupied are replaced by null bytes.

Inserting an Instruction within a Program. If there are not already null bytes available where you want to insert a new instruction, seven null bytes are inserted and all subsequent instructions bumped down seven bytes in memory. The new instruction replaces inserted null bytes and, if the new instruction requires fewer than seven bytes, the rest of the inserted null bytes remain.

Program Lines That Are Numbers. The computer places a null byte before a string of bytes representing a number. This is done in case the previous program line is also a number. The null byte acts as a spacer between the two program lines so they won't be misinterpreted as a single number.

Packing

When your program is complete, the only useful null bytes are those separating sequential program lines that are both numbers. To eliminate unneeded null bytes, execute GTO . . . When memory is packed, bytes within all programs move up in program memory to replace unneeded null bytes. (User-keyboard memory is also packed as described below.) Main memory is packed when:

- You execute PACK.
- You execute GTO . . .
- You clear a program by executing CLP.
- There are not enough uncommitted registers available to complete an operation that requires them. Such operations are: increasing the data register allocation, entering a program line, or assigning a function to a key.

User Keyboard Memory

When you assign a function to a key, that information is stored in User keyboard memory. An assignment for either a function or global label in a plug-in module is also stored in User keyboard memory. However, when you assign a global label listed in catalog 1 to a key, that information is *not* stored in User-keyboard memory, but rather with that global label in program memory.

A register can hold two assignments. The first assignment requires one register; the second assignment fits with the first assignment in that register. Similarly, each odd-numbered assignment adds another register to the User keyboard memory, and each even-numbered assignment fills out the register.

An assignment is cancelled when you assign a different function to the same key, or if you explicitly cancel the assignment as explained in section 1. If both assignments in a register have been cancelled and main memory is packed, that register becomes an uncommitted register.

Data Register Memory

Allocation

Changing the Allocation. The SIZE function allocates main memory registers to data storage. Decreasing the number of registers loses the data in the largest-numbered registers.

You can change the allocation to data storage by executing SIZE and then specifying the number of registers to be allocated. This function is not programmable.

Registers Above R_{99}

If you allocate more than 100 registers to data storage, registers whose addresses exceed 99 can be accessed *only* by indirect addressing. To emphasize this distinction, this manual shows three-digit addresses in parentheses: $R_{(120)}$, for example.

Data Register Operations

Store and Recall

There are two sources/destinations for the data in data registers: the stack registers and the Alpha register. The functions that move data between the stack registers or the Alpha register and the data registers in main memory are described in this section.

Specifying a Register as a Parameter. Most data register functions access just one register, whose address must be specified as a parameter. You can specify a register in several ways.

- For R_{00} through R_{99}, key in the two-digit address.
- For convenience, R_{01} through R_{10} can be specified with a single key in the top two rows.
- For the stack or LAST X registers, press · followed by X, Y, Z, T, or L.
- For any register to be addressed indirectly, press and then specify the address of the indirect register by one of the means above.

Store. To copy data from the X-register into a data register, press STO and then specify the destination register. The X-register is unchanged; the data previously in the data register are lost.

Recall. To copy data from a data register into the X-register, press RCL and then specify the source register. The contents of the source register are unchanged. If stack lift was disabled, the recalled data replace the contents of the X-register; otherwise the stack is lifted.

Alpha Store. To copy the six leftmost characters from the Alpha register into a data register, press ASTO and then specify the destination register. The contents of the Alpha register are unchanged and the data previously in the destination register are lost.

- A punctuation mark counts as one of the six characters.
- A string of digits in the Alpha register is not a number. If you store Alpha digits in a register, the contents *appear* to be a number, but you can't perform numeric operations on those contents.
- Copying data from the Alpha register to the X-register by using ASTO is *not* like RCL—that is, the stack does not lift and so the previous contents of the X-register are lost.

To copy more than six characters into a data register you must alter the contents of the Alpha register before repeating [ASTO] (or you will copy the same characters again). To remove the six characters you already copied, execute [ASHF] (*Alpha shift*). The six leftmost characters are shifted out of the Alpha register.

Alpha Recall. To copy data from a data register into the Alpha register, execute [ARCL] and then specify the source register. The contents of the source register are unchanged, the data are *appended* to the contents of the Alpha register, and character entry is activated. If you want the copied data to start a new message, execute [CLA] before recalling that data.

For an example of the use of [ASTO] and [ARCL] in a program, refer to the WORD GUESSING GAME program in section 9.

Register Arithmetic

Register arithmetic enables you to combine a number in the X-register and a number in a data register without recalling the stored number to the stack.

- Executing [STO] [+] *nn* adds the number in the X-register to the number in R_{nn}, and then stores the sum in R_{nn}.
- Executing [STO] [−] *nn* subtracts the number in the X-register from the number in R_{nn}, and then stores the difference in R_{nn}.
- Executing [STO] [×] *nn* multiplies the number in the X-register by the number in R_{nn}, and then stores the product in R_{nn}.
- Executing [STO] [÷] *nn* divides the number in the X-register into the number in R_{nn}, and then stores the quotient in R_{nn}.

As with [STO], the original number in Rnn is lost and the number in the X-register is unchanged. This allows you to reuse a constant in the X-register without executing [LASTx].

Exchange

Note that [STO] and [RCL] duplicate one number and lose another. To move numbers *without* duplicating or losing any data, execute [X<>] and specify the register whose contents you want to exchange with the X-register.

Clearing Registers

To clear a single register, store zero in that register. To clear all data registers, execute [CLRG].

Part II
Programming in Detail

Section 5

Programming Basics

Contents

Loading a Program

Keying In a Program

1. Press PRGM to select Program mode.
2. Press GTO . . to set the computer to the bottom of program memory.
3. Press LBL followed by a global label.
4. Key in instuctions using the Normal, User, and Alpha keyboards just as you would in Execution mode.
5. Press GTO . . to complete the program (optional).

Pressing GTO · · has the following effects:

- Main memory is packed, ensuring that the maximum number of registers will be available for the next program or key redefinition.
- An END instruction is inserted to complete the last program, creating a null program (consisting of the permanent **.END.**) at the bottom of program memory. (One reason to press GTO · · after loading a program is to give the program its own END instruction.)
- The computer is positioned to this null program and displays **00 REG *nnn*** where *nnn* indicates the number of registers available for a new program. As you key in instructions, they become a new program at the bottom of program memory.

The number of available registers also appears with the permanent **.END.**. If the last program line is displayed, you can press SST to see **.END. REG *nnn***. To then continue adding instructions, simply key them in. To then review your program:

- Press SST to set the computer to the first line of your program.
- Press BST to set the computer back to the last line keyed in.

Copying a ROM Program

If you want to alter a program that is in ROM (*read-only memory*) such as an application module, you must first copy the program into program memory. To do so, execute COPY and specify any global label in the ROM program. A copy of the ROM program is then added to the bottom of program memory.

Enlarging Program Memory

If there is not enough room in memory to store an instruction being added or a program being copied, the computer displays **PACKING** and then **TRY AGAIN**. If you try again but **TRY AGAIN** appears a second time, do one or more of the following steps to increase the number of registers available for program instructions:

- Allocate fewer registers to data storage using SIZE.
- Delete complete programs using CLP.
- Cancel User-keyboard assignments other than global labels listed in catalog 1, then execute PACK or GTO · ·.

Executing a Program

You can execute a program by ensuring that the computer is in Execution mode and then performing one of the following:

- Pressing [XEQ] and specifying a global label in the program. Execution starts with that global label
- Assigning a global label to a key and then pressing that key when the User keyboard is active. Execution starts with that global label.
- Positioning the computer to the beginning of the program and then pressing [R/S]. Execution starts with the current program line.
- Positioning the computer to the beginning of the program and then pressing [SST]. Only the current program line is executed and the computer is positioned to the next program line. This single-step execution is most useful when you're trying to isolate an error in a program. By checking the result after each instruction is executed, you can find where the program goes wrong.
- Positioning the computer to the beginning of the program, setting flag 11, and turning off the computer. When you next turn it on, the computer automatically runs the program starting at the current program line.

The **PRGM** annunciator appears in the display while a program is running. Unless a function like [AVIEW] displays a message, the program execution indicator (⊁) appears in the display; each time the program executes a label, the program execution indicator moves one position to the right.

Program Lines

In Program mode the computer displays one line of program memory at a time. Lines are created automatically as you key in instructions. Each line is assigned a number to indicate its position within the program, and each separate program has its own set of line numbers. Each line contains a complete instruction consisting of:

- A function.
- An Alpha string of up to 15 characters.
- A complete number of up to 10 digits, or up to 10 digits plus a two-digit power of 10.

For details about keying in Alpha strings and numbers, refer to section 1, "The Keyboard and Display."

In a displayed program line, the symbol ᵀ indicates that the characters following comprise an Alpha string or (if preceded by **XEQ**, **GTO**, or **LBL**) a global label. To enter a function into a program line using its Alpha name you must press [XEQ] first. Otherwise, the computer won't recognize the Alpha characters as a function name, but will treat them as an Alpha string and enter them into the Alpha register when it executes that program line.

Nonprogrammable Operations

The following operations are not programmable, but some can be accomplished by other means. Programmable alternatives are shown in parentheses following the nonprogrammable operation.

- Destructive operations:

  [←], [DEL].

  [CLP].

- Positioning operations:

  [GTO] [·] [·], [GTO] [·], [SST], [BST].

- All catalogs.
- Toggle keys:

  [ON] (but [OFF] is programmable).

  [PRGM].

  [USER] (but a program can set or clear flag 27).

  [ALPHA] (but [AON] and [AOFF] are programmable).

- Other nonprogrammable functions:

  [COPY], [ON], [PACK], [R/S] (to run a program).

  [ASN].

  [SIZE].

Positioning Within Program Memory

There are several methods of positioning the computer within Program memory. Some enable you to go to any program in memory (that is, to any global label) while others enable you to go to any line within a program. Some work only in Execution mode, while others work only in Program mode. Only one function, [GTO] [·], can do either job in either mode.

Using [GTO] [·]

In Program or Execution mode:

- To position the computer to any global label, press [GTO] [·] and specify the global label. The search for the label begins with the last global label (as listed by catalog 1) and proceeds upward in memory, stopping at the first matching label encountered.
- To position the computer to line number *nnn* of the current program, press [GTO] [·] *nnn*. If *nnn* exceeds the line number of the last line in the program, the computer is positioned to the last line.

  To position the computer to line 1*nnn* (the line number exceeds 999), press [GTO] [·] [EEX]. When the computer displays **GTO .1___**, key in *nnn*.

Using Catalog 1

In a few cases you can't use [GTO] [·] to position the computer to the desired program. Such cases include:

- The program contains no global labels.
- The desired label is duplicated later in program memory, so that [GTO] [·] always finds the duplicate label first.
- You've forgotten the exact spelling of the global label.

You can position the computer to any global label or [END] statement in program memory using catalog 1 in Program or Execution mode as follows:

1. Press [CATALOG] 1 to display all global labels and [END] statements in program memory.
2. To speed up the listing, press any key other than [ON] or [R/S].
3. Press [R/S] to halt the listing at the desired global label or [END] statement.
4. To display the next item or the previous item in the catalog listing, press [SST] or [BST].
5. Press [←] to position the computer to the displayed item.

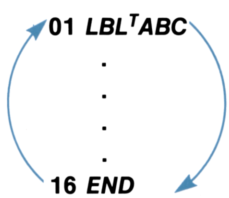
If a program doesn't contain any global labels, follow the five steps above to position the computer to the program's [END] statement. (When two [END] statements appear sequentially, the second [END] statement belongs to a program without global labels.) You should then insert a global label at the start of the program by pressing [GTO] [·] 000 and then (in Program mode) keying in the global label.

Single Step and Back Step

In Program mode you can position the computer to the next program line or to the previous program line by pressing [SST] or [BST].

- Press [SST] to position the computer to the next program line. If the current program line is the last program line, pressing [SST] positions the computer to the first program line (line 01).
- Press [BST] to position the computer to the previous program line. If the current program line is the first program line (line 01), pressing [BST] positions the computer to the last program line.

Pressing [SST] when the computer is positioned at the bottom of the program moves the calculator back to the beginning of this program.



Pressing [BST] when the computer is positioned at the top of the program moves the calculator to the end of this program.

Other Methods

When the computer is in Execution mode you can position it within program memory by using any of the following methods:

Positioning to a Global Label. Press GTO and specify the global label.

Positioning to an Assigned Global Label. If a global label is assigned to a key, hold down that redefined key while you press R/S, and then release the redefined key.

Positioning to a Numeric Label in the Current Program. To position the computer to LBL *nn*, press GTO *nn*. The computer searches for LBL *nn* (as described in section 7, "Branching") and stops at the first matching label encountered.

Positioning to the Top of the Current Program. To position the computer to line 00, press RTN. The computer displays **00 REG *nnn***, indicating that there are *nnn* registers available; if you key in an instruction, that instruction becomes line 01. This is the easiest way to add an instruction at the very beginning of a program.

Editing a Program

All program editing—both deleting and inserting instructions—takes place in Program mode.

Deleting Instructions

Deleting Single Lines. To delete a single instruction, position the computer to the desired program line, and then press ←. That program line is deleted, the computer is positioned to the previous line, and the line number of each subsequent instruction is reduced by one.

When deleting a few lines, start with the last (largest-numbered) line to be deleted. In the example below, suppose that you want to delete lines 02 through 04. At left, the computer is positioned to line 04. Pressing ← deletes line 04 and positions the computer to line 03; pressing ← again deletes line 03 and positions the computer to line 02; and pressing ← a third time deletes line 02 and positions the computer to line 01.

	01 LBL^{T}AREA			
	02 X↑2	01 LBL^{T}AREA		
	03 PI	02 X↑2	01 LBL^{T}AREA	
Current Program Line (displayed)	04 *	03 PI	02 X↑2	01 LBL^{T}AREA
	05 END	04 END	03 END	02 END

Deleting Multiple Lines. To delete a long sequence of instructions:

1. Position the computer to the first (smallest-numbered) line to be deleted.
2. Execute [DEL] (*delete*).
3. Specify the number of lines to be deleted. To delete more than 1000 lines, press [EEX]. When the computer displays **DEL 1_ _ _**, key in the remaining three digits.

In the previous example, lines 02, 03, and 04 are deleted one by one. Alternatively you could position the computer to line 02 and execute [DEL] 003. This deletes lines 02, 03, and 04, leaving the computer positioned to the previous line (line 01). The line number of each subsequent instruction is reduced by three.

If you execute [DEL] *nnn* when there are fewer than *nnn* program lines following the current line, the current line and all subsequent lines *except* [END] are deleted.

Inserting Instructions

To insert an instruction in a program, position the computer to the existing line that you want the new line to follow, and then key in the new instruction. (If you just deleted an instruction using [←] and now you're replacing it, the computer is already properly positioned.) The new instruction becomes the current line, and the line number of each subsequent instruction is increased by one.

When inserting several instructions, start with the first (smallest-numbered) line to be inserted. Suppose that you want to restore the instructions deleted in the previous example. At left, the computer is positioned to line 01. As each instruction is keyed in, it is inserted after the previous current program line and becomes the new current program line.

				01 LBL^{T}AREA
			01 LBL^{T}AREA	02 X↑2
		01 LBL^{T}AREA	02 X↑2	03 PI
Current Program Line (displayed)	01 LBL^{T}AREA	02 X↑2	03 PI	04 *
	02 END	03 END	04 END	05 END

Clearing Programs

The nonprogrammable function [CLP] *(clear program)* will clear one program.

Execute [CLP] and specify any global label in the program to be cleared. The computer then:

1. Searches upward through program memory for the specified global label, beginning with the last global label (as listed by catalog 1).
2. Deletes all instructions (line 01 through [END]) in the first program encountered that contains the specified global label.
3. Packs main memory.

Executing [CLP] and pressing [ALPHA] [ALPHA] without specifying a global label clears the current program.

Programming Examples

Section 9 contains five sample programs. The five programs are:

- RPN PRIMER, to aid in understanding and using the stack;
- FINANCIAL CALCULATIONS, converts your HP-41 into a financial calculator;
- CURVE FITTING, fits data to one of four curves: straight line, exponential, logarithmic or power;
- WORD GUESSING GAME, a version of the word game "hangman;"
- BLACKJACK, a simple version of the card game.

These programs demonstrate many of the basics described in this section and the other sections in this manual.

Flags

Contents

Introduction

A flag has only two states, *set* and *clear*. These states can be interpreted as "on/off" (like a switch), as "yes/no" (like a decision), or as "1/0" (like a binary digit, or bit). The computer has 56 flags, grouped into three types according to use.

User Flags. You can both test and alter user flags. Their status is altered only by your instructions.

Control Flags. You can both test and alter control flags. The computer resets some control flags to default status each time you turn it on, and alters some in the course of operation.

System Flags. You can test system flags but you can't alter them.

You can set and clear flags 00 through 29, which are the user and control flags.

- To set a flag, press [SF] and then specify the flag number.
- To clear a flag, press [CF] and then specify the flag number.

You can test flags 00 through 55 by pressing [FS?] and then specifying the flag number. The display shows **YES** if the flag is set, or **NO** if the flag is clear. Flag tests like [FS?] are used primarily to control program execution, as described in section 7, "Branching."

Types of Flags

User Flags (00 through 10)

The user flags are solely for your own use; what they mean depends entirely on how you use them. For example, a program can ask whether the user wants English or metric units, and then store the user's response as the status of one user flag. Afterwards, whenever the program needs to check which units to use, it can test that user flag.

The state of each user flag is maintained by Continuous Memory. Once you set or clear a user flag, its status is fixed until you alter it. When any of the first five flags is set, the corresponding annunciator (**0**, **1**, **2**, **3**, or **4**) appears in the display.

Two programs in section 9, RPN PRIMER and BLACKJACK, use the user flags. RPN PRIMER uses flag 5 to represent the status of the stack: enabled or disabled. The BLACKJACK program uses flags 6 through 9 to represent various playing situations. (The meaning of each flag is listed after the program listing.)

Control Flags (11 through 29)

The control flags have specific meanings to the computer, listed below. The status of these flags represent certain operating conditions and options. You can alter these flags to indicate your choice of options; the computer alters some of these flags to indicate conditions, which you can then check by testing the flags.

Flag 11: Automatic Execution. Flag 11 allows a program to run automatically. If you set flag 11 before you turn off the computer, the following will happen when you next turn it on:

- A tone sounds.
- Program execution begins from the current program line.
- Flag 11 is cleared.

Flags 12 through 20: External Device Control. These flags direct the operation of external devices that are controlled by the computer. All flags for external device control are cleared each time you turn on the computer. The precise meaning of these flags depends on the particular devices that are present; refer to the appropriate manuals for details.

Flag 21: Printer Enable. Flag 21 allows your program to control how functions like $\boxed{\text{VIEW}}$ and $\boxed{\text{AVIEW}}$ are executed, depending on whether an output device is present. For details, refer to appendix C, "Printer Operation."

Flags 22 and 23: Data Input. These flags allow a program that prompts for input to determine the the user's response.

- Flag 22 is set when numbers are keyed into the X-register.
- Flag 23 is set when characters are keyed into the Alpha register.

These flags are cleared automatically only when you turn on the computer. If you intend to test these flags, you should clear them before prompting for the response.

The FINANCIAL CALCULATIONS program and BLACKJACK program in section 9 use flag 22.

Flags 24 and 25: Error Ignore. Normally, an error condition halts program execution. These flags allow you to avoid unnecessary program halts and to use error conditions as a programming tool.

- If flag 24 is set, the computer ignores *all* **OUT OF RANGE** errors. This error normally results from any calculation (except statistical accumulations) that produces a number x such that $|x| > 9.999999999 \times 10^{99}$. If flag 24 is set, $\pm 9.999999999 \times 10^{99}$ is returned as an approximation to the correct answer, and program execution continues.

  Flag 24 is cleared each time you turn on the computer. Once you set flag 24, it remains set until you explicitly clear it or turn off the computer. If you want to branch to your own error subroutine rather than use $\pm 9.999999999 \times 10^{99}$ as an approximation, use flag 25.

- If flag 25 is set, the computer ignores *only one* error of any kind and then clears flag 25. The command that caused the error is not executed. Flag 25 is cleared each time you turn on the computer.

  If both flags 24 and 25 are set, an **OUT OF RANGE** result will be handled by flag 24—flag 25 will *not* be cleared. Note that if flag 25 is set but not flag 24, an **OUT OF RANGE** result will *not* cause $\pm 9.999999999 \times 10^{99}$ to be placed in the appropriate register.

  You can detect an error by setting flag 25 just before a command and, just after the command, testing if flag 25 was cleared. (Generally you should test *and clear* flag 25—it's dangerous to ignore unanticipated errors.) This enables a program to branch rather than stop execution in case of an error.

Flag 26: Audio Enable. When flag 26 is set, [BEEP], [TONE], alarms, and the stopwatch produce audible tones. Flag 26 is set each time you turn on the computer. (This is the only control flag whose default status is *set*.) You can silence the computer by clearing flag 26.

Flag 27: User Keyboard. Flag 27 is set when the User keyboard is active—that is, when the **USER** annunciator is displayed. A program can check or alter this flag exactly as you can check the annunciator or press [USER]. Flag 27 is maintained by Continuous Memory.

Flags 28 and 29: Display Punctuation. These flags control the use of periods and commas in numeric displays and are maintained by Continuous Memory. For details, refer to "Display Format" in section 1.

RPN PRIMER, CURVE FITTING, WORD GUESSING GAME, and BLACKJACK, in section 9, clear flag 29 so that no separator marks will be displayed.

System Flags (30 through 55)

The system flags are primarily for internal use by the computer; their utility to the user is limited. You can test system flags, but several always test *clear*. You can't directly alter individual system flags, but you can save and restore the status of those that represent user options. Listed below are ways you can use some of the system flags.

Flags That Represent Options. Some external devices controlled by the computer use system flags to represent options relating to those devices; refer to the appropriate manuals for details. The following system flags represent options in the computer:

- Flags 36 through 39 represent the number of displayed digits, described in section 1.
- Flags 40 and 41 represent the display format, described in section 1.
- Flags 42 and 43 represent the angular mode, described in section 3.

Flags That Represent Conditions. The following flags provide information that is useful for some programs:

- Flag 44 is set when ON (*continuous on*) is executed.
- Flag 48 is set when the Alpha keyboard is active—that is, when the **ALPHA** annunciator is displayed.
- Flag 49 is set (and the **BAT** annunciator is displayed) when battery power is low. A long-running program can occasionally test flag 49 and execute OFF if flag 49 is set. Otherwise, if a program continues to run when battery power is low, the memory contents of the computer can be affected.
- Flag 50 is set when a message is displayed.
- Flag 55 is set if a printer is present. This flag works with flag 21 (Printer Enable); their interaction is described in appendix C, "Printer Operation."

Summary of Flag Status

The chart on the next page indicates flag status when Continuous Memory has been cleared ("Reset") and whenever you turn on the computer ("Turn-On"). In addition to *clear* and *set*, there are two flag states coded as follows:

M = Maintained by Continuous Memory.
? = Dependent on other conditions.

Summary of Flag Status

Flag Number	Flag Name	Status at Reset,	at Turn-On
00-10	User Flags	Clear	M
11	Automatic Execution	Clear	Clear
12-20	External Device Control	Clear	Clear
21	Printer Enable	?	?
22	Numeric Data Input	Clear	Clear
23	Alpha Data Input	Clear	Clear
24	Range Error Ignore	Clear	Clear
25	Error Ignore	Clear	Clear
26	Audio Enable	Set	Set
27	User Keyboard	Clear	M
28	Display Puncuation	Set	M
29	Separator Mark	Set	M
36	Number of Digits	Clear	M
37	〃	Set	M
38	〃	Clear	M
39	〃	Clear	M
40	Display Format	Set	M
41	〃	Clear	M
42	Angular Mode	Clear	M
43	〃	Clear	M
44	Continuous On	Clear	Clear
48	Alpha Keyboard	Clear	Clear
49	Low Battery	?	?
50	Message	Clear	Clear
55	Printer Existence	?	?

Section 7

Branching

Contents

Introduction

Branching occurs whenever program execution jumps to an instruction other than the next program line—that is, whenever program steps are not executed sequentially. Two types of functions cause branching:

- Executing [GTO] *label* or [XEQ] *label* causes program execution to branch to the specified label.
- Executing a flag test, comparison, or loop control function can cause program execution to skip the next program line, depending on whether a certain condition is true.

Often these two types of functions are used together: a flag test can be followed by [GTO] *label*, so that the status of the specified flag determines whether program execution branches to the specified label. This section describes the use of [GTO] first, [XEQ] next, conditional functions (flag tests and comparisons) next, and looping last. For examples of branching, refer to the programs in section 9.

Branching to a Label

The only purpose of labels is to serve as targets for branching instructions. The two basic types of labels are global labels, which can be accessed from any program in program memory, and local labels, which can be accessed only from inside their own program. Any label other than a local Alpha label can be specified indirectly as well as directly.

Global Labels

Global labels consist of up to seven Alpha characters including digits. Commas, periods, and colons are not allowed. Single letters from A through J and from a through e are called local Alpha labels and can't be used as global labels. However, other single letters or digits are legal global labels. Global labels require four bytes of program memory plus one additional byte for each character.

Programs are identified by their global labels. Functions that act on entire programs (like CLP) require a global label to specify the program. At the same time, a global label also identifies a particular line in a program—namely itself. You can branch to different parts of a program from outside that program if it contains several global labels; any one of these global labels can serve to identify the entire program.

Global Label Searches

When the computer executes GTO followed by a global label, it first searches within program memory. The search begins with the last global label (as listed by catalog 1) and proceeds upward through program memory, stopping at the first label that matches the specified label. The search is in the opposite order from the catalog 1 listing. If there are two global labels using the same characters, the higher label (listed first by catalog 1) is never found because the search always stops at the lower label.

If the computer reaches the top of program memory without finding the specified label, it then searches in catalog 2. If a program in a plug-in module or peripheral device includes the specified global label, execution is transferred to the module or device and continues from that label.

Local Labels

Local labels are the internal markers in a program, used for branching within the current program. The three types of local labels are described first, followed by how the computer searches for local labels.

Local Numeric Labels. There are two types of numeric labels, one for branching a limited distance and another for branching any distance within a program.

- Labels 00 through 14 are *short-form* numeric labels, requiring only a single byte of program memory. Use them only when the distance in program memory from the GTO instruction to the label is 112 bytes or less.
- Labels 15 through 99 are *long-form* numeric labels, requiring two bytes of program memory. They can be used for branching any distance within a program.

Local Alpha Labels. Local Alpha labels require two bytes of program memory and can be used for branching any distance within a program. They are designed for manual execution: when the User keyboard is active, a local Alpha label is automatically assigned to each key on the top two rows (as described in "The Top Two Rows" in section 1 and demonstrated in the FINANCIAL CALCULATIONS program in section 9). You can then use these keys to execute the corresponding local Alpha labels in the current program.

Local Label Searches

Searches for local labels occur only within the current program. To find a local label, the computer first searches sequentially downward through the current program, starting at the GTO instruction. If the specified label is not found before reaching the end of the program, the computer continues the search from the beginning of the program.

A local label search can consume a significant amount of time, depending on the length of the current program. To minimize the search time, the computer records the distance in program memory from the GTO instruction to the specified local label when the GTO instruction is first executed. This eliminates the search time for subsequent executions of that GTO instruction.

Bytes for a GTO Instruction

The number of bytes of program memory required by a GTO instruction depends on which type of label is specified:

- A GTO instruction specifying a global label of n characters requires $2 + n$ bytes.
- A GTO instruction specifying a long-form numeric label or a local Alpha label requires three bytes.
- A GTO instruction specifying a short-form numeric label or an indirect address requires two bytes.

Calling a Subroutine

A program instruction consisting of XEQ followed by a label is a special type of branch named a *subroutine call.* XEQ *label* and GTO *label* are similar in that:

- Both transfer program execution to the specified label.
- All types of labels that can be specified for GTO can also be specified for XEQ.

A subroutine call is special because of what occurs *after* XEQ has transferred execution to the specified label: the next RTN or END instruction executed will return program execution to the instruction that follows the XEQ instruction, as illustrated below.

Program ABC branches to program DEF, so execution stops when the END instruction is encountered at the end of DEF.

```
01 LBL^T ABC          01 LBL^T DEF
 :                     :
10 GTO^T DEF          40 END
11 STO 01
 :
30 END
```

Program ABC calls program DEF as a subroutine, so execution returns to ABC when the END instruction is encountered at the end of DEF.

```
01 LBL^T ABC          01 LBL^T DEF
 :                     :
10 XEQ^T DEF          40 END
11 STO 01
 :
30 END
```

Using subroutines saves space in program memory. The instructions in the subroutine appear only once, but they can be executed any number of times both within a program and (if the subroutine begins with a global label) from any number of programs.

Either RTN or END causes execution to return to the instruction following the subroutine call. However, END marks the end of the program and thus affects local label searches and functions that act on entire programs; RTN marks only the end of a subroutine within a program. In the following program END terminates the subroutine and RTN terminates program execution. (In practice, there would be no reason to execute lines 22 through 29 as a subroutine because they are executed only once.)

```
01 LBL^T ABC
 :
10 XEQ 00
11 STO 01
 :
20 RTN
21 LBL 00
22 SIN
 :
30 END
```

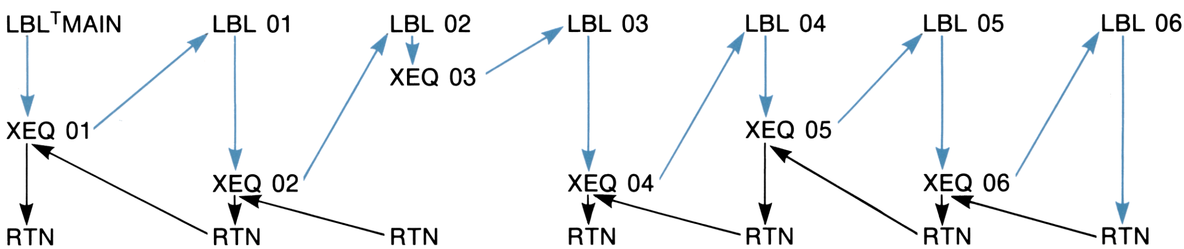
If you call ABC as a subroutine from another program, execution returns to the calling program when 20 RTN is executed. That is, if a program calls a subroutine that calls a second subroutine, the second subroutine is completed and execution returns to the first subroutine; then the first subroutine is completed and execution returns to the calling program.

Alternatively, you can ensure that execution will stop at line 20, even if ABC is called as a subroutine, by entering 20 STOP. Press [R/S] in Program mode to enter a [STOP] instruction.

The Subroutine Return Stack

When an [XEQ] instruction calls a subroutine, the computer remembers the location in program memory of that [XEQ] instruction, so that execution can return there when the subroutine is completed. While the subroutine is being executed, this return location is stored in the *subroutine return stack*. When the subroutine is completed and execution returns to the [XEQ] instruction, the location of the [XEQ] instruction is removed from the subroutine return stack.

Subroutine Limits. When a subroutine calls another subroutine, all pending return locations in the subroutine return stack are "pushed up" in the stack. The subroutine return stack can hold six pending return locations, so the computer can return from subroutines up to six levels deep.



Loss of Subroutine Returns. Pending return locations are lost from the subroutine return stack under the following conditions.

- If there are already six pending return locations in the subroutine return stack when a subroutine is called, the earliest return location is lost from the stack. In this case, program execution never returns to the [XEQ] instruction that called the first subroutine; instead, excution halts when the first subroutine is finally completed because there are no further return locations in the stack.
- All pending return locations are lost when you manually execute a program. Therefore, if you stop a program ABC in the middle of a subroutine and manually execute a program DEF, it will be impossible to resume ABC. DEF need not be a different program from ABC; for example, executing a local Alpha label by pressing a key on the User keyboard clears the subroutine return stack.

Global-Label Subroutine Searches

When the computer executes [XEQ] followed by a global label, it first searches the contents of program memory just as it does for [GTO]. However, if the specified label isn't found in program memory, the next stages of the search caused by [XEQ] differ from the search caused by [GTO]. The order of the complete search caused by [XEQ] corresponds to the numbers of catalogs 1, 2, and 3.

Searching Catalog 1. The search begins with the last global label (as listed by catalog 1) and proceeds upward through program memory, stopping at the first label that matches the specified label. Execution then resumes at that matching label.

Searching Catalog 2. If the specified label isn't found in program memory, the computer then searches catalog 2 for a global label *or function name* that matches the specified label. (Refer to appendix D for a detailed explanation of the contents of catalog 2.) Execution then resumes at that matching label, or the function with the matching name is executed.

Searching Catalog 3. If the specified label isn't found in catalog 2, the computer then searches catalog 3 for a function whose name matches the specified label. If such a function is found, it is executed; otherwise a **NONEXISTENT** error occurs.

Bytes for an [XEQ] Instruction

The number of bytes of program memory required by an [XEQ] instruction depends on which type of label is specified.

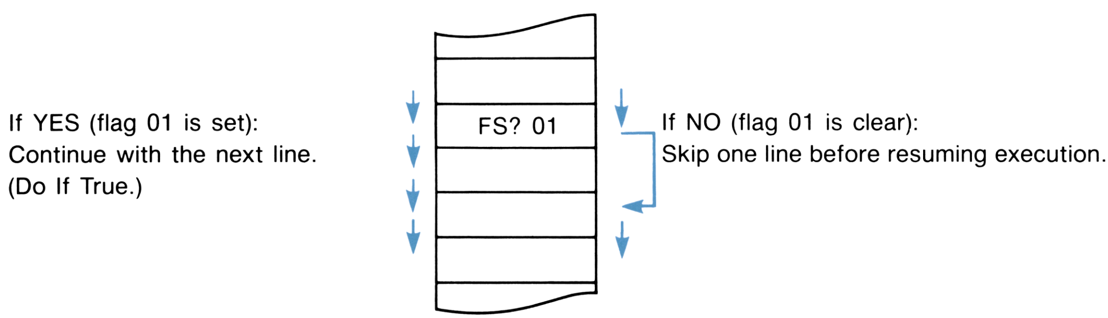
- An [XEQ] instruction specifying a global label of n characters requires $2 + n$ bytes.
- An [XEQ] instruction specifying a local label requires three bytes.
- An [XEQ] instruction specifying an indirect address requires two bytes.

Conditional Functions

Flag tests and comparisons are conditional functions. They express a proposition that is true or false depending on current conditions, and their effect depends on whether the proposition is currently true or false.

- If you manually execute a conditional function, the computer displays **YES** if the proposition is currently true or **NO** if the proposition is currently false.
- If a program executes a conditional function, the result follows the rule: DO IF TRUE. The program line that follows the conditional function is executed if the proposition is currently true, or else is skipped if the proposition is currently false. That is, DO the next instruction IF the proposition is TRUE.

Is Flag 01 Set?



Flag Tests

The following functions can test any flag.

FS? *nn* Is flag *nn* set? (00 ⩽ *nn* ⩽ 55)

FC? *nn* Is flag *nn* clear? (00 ⩽ *nn* ⩽ 55)

Two functions test and then clear a flag. They can't act on system flags (30 through 55) because you can't alter system flags.

FS?C *nn* Is flag *nn* set? Clear flag *nn*. (00 ⩽ *nn* ⩽ 29)

FC?C *nn* Is flag *nn* clear? Clear flag *nn*. (00 ⩽ *nn* ⩽ 29)

Comparisons

Comparing X with Zero. The following five functions compare the number in the X-register with zero:

X<0? X<=0? x=0? X≠0? X>0?

Comparing X with Y. The following five functions compare the number in the X-register with the number in the Y-register.

X<Y? x⩽y? x=y? X≠Y? x>y?

Two of these functions, x=y? and X≠Y?, can compare Alpha data as well as numeric data. Executing any of the three other functions with Alpha data in the X- or Y-register causes an **ALPHA DATA** error.

Looping

A loop is a sequence of instructions that starts with a label and ends with a branch back to that label. The simplest case is an infinite loop such as the following program.

```
01 LBL⊤LOOP
02 BEEP
03 GTO⊤LOOP
04 END
```

Once started, this program would run until the batteries expired. Infinite loops should generally be avoided, but loops that repeat themselves until some condition is met are a powerful programming tool.

Looping Using Conditional Functions

When you want to perform an operation until a certain condition is met but you don't know exactly how many times to repeat the operation, you can create a loop with a conditional function just before the GTO instruction. For example, the following program subtracts one from a number, tests the result, and repeats the loop if the result is positive. As soon as the number is reduced to zero (assuming that the original number was positive), the program exits the loop and beeps.

```
01 LBLTABC
02 1
03 –
04 X>0?
05 GTOTABC
06 BEEP
07 END
```

Loop-Control Functions

When you want to execute a loop a specific number of times, you can use special functions for that purpose instead of the conditional functions in the previous examples. These special functions are ISG *(increment, skip if greater)* and DSE *(decrement, skip if equal)*. Both functions use a control number in a register to control looping. This register can be a data register in main memory, a stack register, or the LAST X register; it can be specified indirectly as well as directly.

The format of the loop-control number is *iiiii.fffcc*, where:

***iiiii* is the current counter value.** Each time ISG or DSE is executed, *iiiii* is incremented (for ISG) or decremented (for DSE) by the value of *cc*. The part *iiiii* can consist of one through five digits.

***fff* is the final counter value.** Each time ISG or DSE increments or decrements *iiiii*, the resulting value of *iiiii* is compared with the value of *fff*. The part *fff* must consist of three digits like 100, 020, or 009.

***cc* is the increment/decrement value.** If *cc* is 00 (or unspecified), the computer uses a default value of 01 instead. If specified, *cc* must consist of two digits like 30 or 03.

When the computer executes ISG, it first increments *iiiii* by *cc*, and then tests if the resulting value of *iiiii* is greater than *fff*. If it is, the computer skips the next instruction.

When the computer executes DSE, it first decrements *iiiii* by *cc*, and then tests if the resulting value of *iiiii* is equal to (or less than) *fff*. If it is, the computer skips the next instruction.

The WORD GUESSING GAME in section 9 uses loop-control numbers and DSE to break a word into letters and store the letters in sequential registers.

Section 8

Alpha and Interactive Operations

Contents

Introduction

This section covers the use of the Alpha register: the interaction between the user and a program. Interaction between the user and a program involves the functions that display a message and the functions that interpret the user's response.

Moving data between the Alpha register and the X-register involves the ARCL and ASTO functions.

Executing ARCL X copies the contents of the X-register into the Alpha register; ASTO X copies six characters from the Alpha register into the X-register. (Digits placed in the X-register by ASTO are characters and cannot be used in computations.) The functions ARCL and ASTO, which in general access data registers, are discussed in section 4, "Main Memory."

Requesting Input

There are several functions and combinations of functions that request input from the user. In comparing the alternatives there are two issues:

1. Is program execution stopped until a response is given, or does execution eventually continue even if no response is given?
2. What types of response are possible?

The alternatives below are described in terms of these two issues.

Using PROMPT

When PROMPT is executed, the computer displays the contents of the Alpha register and stops execution. The displayed message should indicate the type of response that is expected: numeric input, Alpha input, a procedure, a keystroke that the program will interpret, or many other possibilities. Before considering the specific examples below, note that there are only two ways to restart program execution:

- You can press R/S to restart execution beginning with the program line that follows PROMPT; or
- You can branch to a local Alpha label by pressing the corresponding key in the top two rows. Execution resumes at the local Alpha label.

The WORD GUESSING GAME in section 9 uses PROMPT to ask the user for a word and letters.

Using PSE

You can use PSE *(pause)* much like PROMPT, but with the following differences:

- PSE delays execution for slightly less that a second. Keying in a number or Alpha string during a pause causes the pause to be repeated; executing a function halts program execution.
- Normally, PSE displays the X-register. To display a message that is in the Alpha register, either AVIEW or AON must precede PSE.

A string of consecutive PSE instructions allows more time to begin a response. Each time PSE is executed, the **PRGM** annunciator blinks once. Digit and character entry are terminated at the end of each pause; if you key in a few digits, wait for more than a second, and then key in more digits, the two groups of digits will be treated as two separate numbers.

Producing Output

The following functions allow the computer to display a message and generate audible signals.

Using AVIEW

When a program executes AVIEW, the computer displays the contents of Alpha register until CLD *(clear display)* clears the display or another message is displayed. Executing AVIEW might also stop program execution, depending on the status of flags 21 (Printer Enable) and 55 (Printer Existence) as described in appendix C. All the programs in section 9 use AVIEW to display program results.

Using VIEW

To display the contents of R_{nn} without recalling the register's contents to the X-register, execute VIEW *nn*. The register to be viewed can also be specified indirectly. When a program executes VIEW *nn*, the computer displays the contents of R_{nn} until CLD clears the display or another message is displayed. Like AVIEW, VIEW's operation is affected by flags 21 and 55.

Using PSE

PSE can be used to briefly display a message. When PSE is executed, program execution halts for slightly less than a second.

- If the display already contained a message (rather than the program execution indicator), this message remains displayed.
- If there was no message in the display and the Alpha keyboard is active, the contents of the Alpha register are displayed.
- Otherwise, the contents of the X-register are displayed.

Using TONE and BEEP

Executing TONE *n* produces a single audible tone with a pitch specified by the value of n. The lowest pitch is produced when $n = 0$, the highest when $n = 9$.

Executing BEEP produces a fixed sequence of four tones.

Section 9

Sample Programs

Contents

Introduction

This section contains five programs to help you understand programming techniques on the HP-41. These programs are useful in their own right, but they also have been referenced throughout the manual to demonstrate a function or technique. The five programs are:

1. RPN PRIMER—An aid to understanding and using RPN logic, by illustrating the four stack registers. It will help when you are learning how you can use the stack in calculations and programs.

2. FINANCIAL CALCULATIONS—Converts your HP-41 into a financial calculator. The program, aside from being useful, demonstrates the use of local alpha labels and a procedure known as interchangeable solutions, whereby the HP-41 knows whether to store the value that is in the X-register or calculate a new value.
3. CURVE FITTING—Four curve fitting routines: straight line, exponential, logarithmic or power curve. The program demonstrates the use of indirect addressing to determine how to process the data, then uses one routine to calculate a, b and R^2.
4. WORD GUESSING GAME—Demonstrates the use of the alpha functions [AOFF], [AON], [ARCL], [ASHF], [ASTO], and [AVIEW]. Included are routines for breaking a word into letters and putting a word back together, using [DSE], [ISG] and indirect addressing.
5. BLACKJACK—A simple version of the card game. Included in this program is a random number generator.

The programs also demonstrate the use of flags, prompting for information, labeling your results with alpha labels, branching, storing and recalling information, and writing a program based on an equation.

Each program includes instructions on running the program, examples of using the program, and the program listing of the steps that you must key in to the HP-41. (These listings include comments about the steps.) The conventions used in these instructions, examples and program listings are the same as the conventions inside the front cover and used throughout the manual.

The instructions for running each program are listed in a five column table. The first column, labeled **Step**, is the instruction step number. Column two is the **Instruction** column, which gives instructions and comments concerning the operations to be performed.

The **Input** column specifies the input data, or appropriate alpha response to a prompt. The **Function** column specifies the keys to be pressed after keying in the input data. The last column, **Display**, shows all prompts and results that appear in the display.

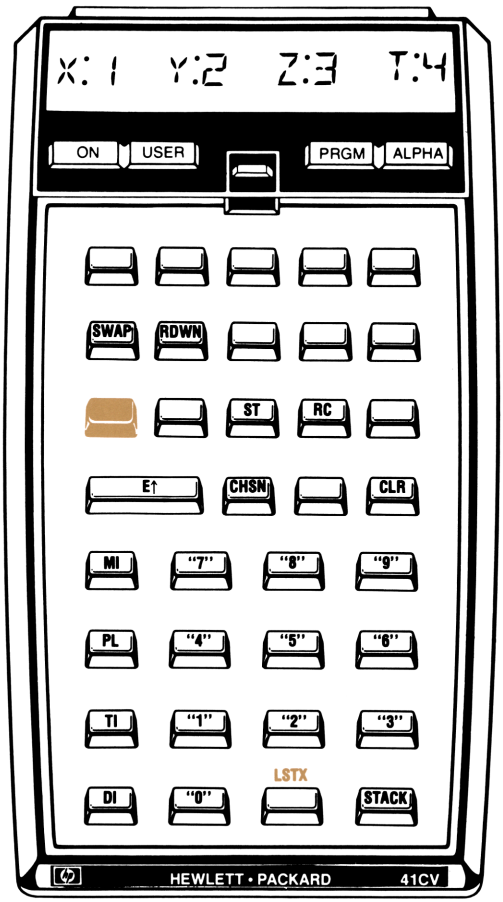
Above and at the right of the instruction table is a box specifying the minimum size and the display format expected by the program. The *HP-41CV Owner's Manual* tells you how to use [SIZE] and [FIX].

These programs, and five more, are also available in the Standard Applications Module (part number 00041-15001). Bar code for these programs is included in the *HP 82153A Wand Owner's Manual.*

RPN PRIMER

This program is an aid to understanding and using RPN, the logic system in the HP-41. All four registers of the operational stack are visible simultaneously so that the effect on the stack of a given keystroke sequence can be seen rather than inferred. The functions in the program should be assigned as shown on the keyboard below. These functions all exit to a routine that displays the stack. You can observe the effect on the stack of any function by executing the function, then the routine STACK. The only operational differences between this redefined calculator and the actual one are that only single-digit numbers can be keyed in and that [STO] and [RCL] address only a single register (thus requiring no address).

Running RPN PRIMER



STATUS: SIZE 010, FIX 02

Step	Instructions	Input	Function	Display
1	Set status (above) and key in the program (pages 85–88).			
2	Assign the routines to the following keys and activate the User keyboard.* These User assignments result in the keyboard shown on the previous page. SWAP [X<>Y] 9 9 ST [STO] 8 8 RDWN [R↓] 7 7 E↑ [ENTER↑] 6 6 RC [RCL] 5 5 CLR [←] 4 4 CHSN [CHS] 3 3 PL [+] 2 2 MI [−] 1 1 MU [×] 0 0 DI [÷] LSTX [LASTX] STACK [R/S]			
3	Press desired keystroke sequence and watch stack contents change.			
4	The functions RUP and CLSTK are obtained by: and: (or you could also assign these functions to keys).		[XEQ] [ALPHA] RUP [ALPHA] [XEQ] [ALPHA] CLSTK [ALPHA]	

* To assign a function, say SWAP, to a key, say the [X<>Y] key: [ASN] [ALPHA] SWAP [ALPHA] [X<>Y].

Example 1:

Evaluate the expression

$$\frac{(2 + b)\,b}{8 - b}$$

for $b = 3$

Keystrokes	Display	
CLSTK	X:0 Y:0 Z:0 T:0	
2	X:2 Y:0 Z:0 T:0	
ENTER↑	X:2 Y:2 Z:0 T:0	After an ENTER↑, the stack does not lift when new data is keyed in.
3	X:3 Y:2 Z:0 T:0	
+	X:5 Y:0 Z:0 T:0	
LASTX	X:3 Y:5 Z:0 T:0	
×	X:15 Y:0 Z:0 T:0	
8	X:8 Y:15 Z:0 T:0	
LASTX	X:3 Y:8 Z:15 T:0	
−	X:5 Y:15 Z:0 T:0	
÷	X:3 Y:0 Z:0 T:0	

Example 2:

Without disturbing the above results, compute

$$\frac{2 + 4\,(9 - 7)}{6 - 4}$$

Keystrokes	Display
9	X:9 Y:3 Z:0 T:0
ENTER↑	X:9 Y:9 Z:3 T:0
7	X:7 Y:9 Z:3 T:0
−	X:2 Y:3 Z:0 T:0
4	X:4 Y:2 Z:3 T:0
×	X:8 Y:3 Z:0 T:0
2	X:2 Y:8 Z:3 T:0
+	X:10 Y:3 Z:0 T:0
6	X:6 Y:10 Z:3 T:0
ENTER↑	X:6 Y:6 Z:10 T:3

Keystrokes	Display	
4	X:4 Y:6 Z:10 T:3	
[−]	X:2 Y:10 Z:3 T:3	
[÷]	X:5 Y:3 Z:3 T:3	Notice that the answer remaining from Example 1 did not cause a difficulty in Example 2.

Example 3:

Convert the complex number 3 + 4i to polar form.

Keystrokes	Display	
4	X:4 Y:5 Z:3 T:3	
[ENTER↑]	X:4 Y:4 Z:5 T:3	
3	X:3 Y:4 Z:5 T:3	
[R→P]	5	
[STACK]	X:5 Y:53 Z:5 T:3	Remember that [STACK] is assigned to [R/S].

Program Highlight

One especially useful function in this program is the display routine STACK (lines 57–67). You might like to keep it handy to view the entire stack from time to time as you solve your own problems.

Program Listing

```
01♦LBL "CLSTK"
02 CLST
03 GTO 14
```

Lines **01 through 03** clear the stack.

```
04♦LBL "1"
05 FS?C 05
06 CLX
07 1
08 GTO 14
```

Line **05** checks if lift is disabled (if flag 5 is set). If it is, line **06** clears the X-register. If not, line **06** is skipped. Line **07** inputs a 1.

```
09♦LBL "2"
10 FS?C 05
11 CLX
12 2
13 GTO 14
```

Lines **09 through 13** input a 2.

```
14♦LBL "3"
15 FS?C 05
16 CLX
17 3
18 GTO 14
```

Lines **14 through 18** input a 3.

```
19♦LBL "4"
20 FS?C 05
21 CLX
22 4
23 GTO 14
```

Lines **19 through 23** input a 4.

```
24♦LBL "5"
25 FS?C 05
26 CLX
27 5
28 GTO 14
```

Lines **24 through 28** input a 5.

```
29♦LBL "6"
30 FS?C 05
31 CLX
32 6
33 GTO 14
```

Lines **29 through 33** input a 6.

```
34♦LBL "7"
35 FS?C 05
36 CLX
37 7
38 GTO 14
```

Lines **34 through 38** input a 7.

```
39♦LBL "8"
40 FS?C 05
41 CLX
42 8
43 GTO 14
```

Lines **39 through 43** input an 8.

```
44♦LBL "9"
45 FS?C 05
46 CLX
47 9
48 GTO 14
```

Lines **44 through 48** input a 9.

```
49♦LBL "0"
50 FS?C 05
51 CLX
52 0
53 GTO 14
```

Lines **49 through 53** input a 0.

```
54♦LBL 13
55 CF 05
```

Enable stack lift by clearing flag 05.

```
56♦LBL 14
57♦LBL "STACK"
58 "X:"
59 ARCL X
60 "⊦ Y:"
61 ARCL Y
62 "⊦ Z:"
63 ARCL Z
64 "⊦ T:"
65 ARCL T
66 AVIEW
67 RTN
```

Lines **57 through 67** display the stack.

```
68♦LBL "E↑"
69 SF 05
70 ENTER↑
71 GTO 14
```

Lines **68 through 71** disable stack lift by setting flag 05.

```
72♦LBL "RDWN"
73 RDN
74 GTO 13
```

Lines **72 through 74** roll down the stack.

```
75♦LBL "SWAP"
76 X<>Y
77 GTO 13
```

Lines **75 through 77** swap X and Y.

```
78♦LBL "RUP"
79 R↑
80 GTO 13
```

Lines **78 through 80** roll up the stack.

```
81♦LBL "PL"
82 +
83 GTO 13
```

Lines **81 through 83** add the contents of the X- and Y-registers.

```
84♦LBL "MI"
85 -
86 GTO 13
```

Lines **84 through 86** subtract the contents of X from the contents of Y.

```
87♦LBL "MU"
88 *
89 GTO 13
```

Lines **87 through 89** multiply X and Y.

```
90♦LBL "DI"
91 /
92 GTO 13
```

Lines **90 through 92** divide Y by X.

```
93♦LBL "CLR"
94 SF 05
95 CLX
96 GTO 14
```

Lines **93 through 96** disable stack lift and clear X.

```
97♦LBL "CHSN"
98 CHS
99 GTO 13
```

Lines **97 through 99** change the sign of X.

```
100♦LBL "ST"
101 STO 00
102 GTO 13
```

Lines **100 through 102** store X in R_{00}.

```
103♦LBL "RC"
104 FS?C 05
105 CLX
106 RCL 00
107 GTO 14
```

Lines **103 through 107** check if lift disabled. If it is, clear X first. Line **106** recalls the contents of R_{00}.

```
108♦LBL "LSTX"
109 FS?C 05
110 CLX
111 LASTX
112 GTO 14
```

Lines **108 through 112** get the value in LAST X register.

Registers Used

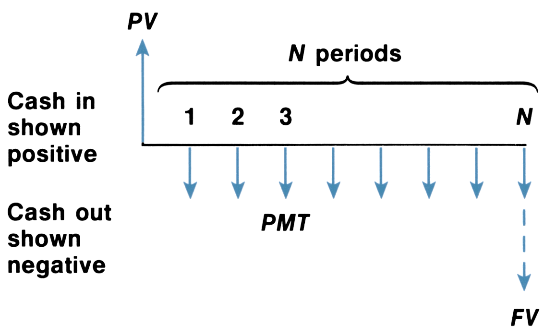
R_{00}: Storage

Flags Used

F05: Set = disable stack; Clear = enable stack	F29: Clear for no separator marks

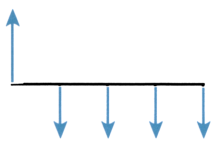
FINANCIAL CALCULATIONS

This program converts your HP-41 into a powerful financial calculator. The program can solve for any of the unknowns relating to a cash flow situation as shown below.

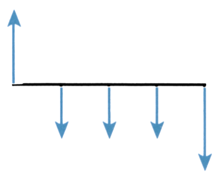


PV = Present Value: the amount loaned, borrowed, invested, etc.
I = Periodic Interest rate.
N = Number of periods.
PMT = Payment amount: the amount paid on a loan or earned on an investment.
FV = Future Value: the amount remaining, accumulated, saved, etc.

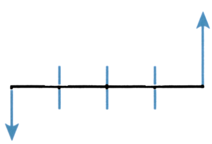
The sketch above shows a standard loan amortization cash flow from the borrower's point of view. From the lender's point of view, PV wuld be shown negative and the PMT stream would be positive. By changing the signs of PV, PMT and FV, different cash flow situations may be realized. Cash flow diagrams for the four basic compound interest problems are presented below along with some of the more common terminology.



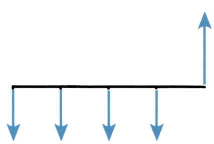
Mortgage
Lease
Direct Reduction Loan
Installment Loan
Amortization
Annuity



Mortgage w/Balloon
Lease w/Buy Back
Lease w/Residual
Annuity



Compound Growth
Savings Account
Appreciation



Savings Plan
Sinking Fund
Pension Fund
Annuity (series of payments)

The five top-row keys (A through E) are used to enter or calculate these financial parameters. If you key in any three parameters, pressing one of the other two keys calculates the corresponding value; if you key in any four parameters, pressing the remaining key calculates its corresponding value. Previously input values can be recalled by pressing RCL followed by the appropriate key. The key sequence a may be used to clear all the registers used by this program. When the registers have been cleared in this manner, the message **N,I,PV,PMT,FV** is put into the display to remind you of the functions of the keys.

For some combinations of values, this program fails to converge to a solution for periodic interest *i*. This effect may be avoided by using a different initial value for *i*.

Reference: More information regarding cash-flow analysis may be found in Grant, E.L. and Ireson, W.G., *Principles of Engineering Economy*, Fourth Edition, The Ronald Press Company, New York, 1964.

Running FINANCIAL CALCULATIONS

STATUS: [SIZE] 010, [FIX] 02

Step	Instructions	Input	Function	Display
1	Set status (above) and key in the program (pages 92–96).			
2	Begin the program:		[FIN]	
3	To clear the finance registers:		[a]	N,I,PV,PMT,FV
4	Store input as desired:			
	number of periods.	N	[A]	N
	periodic interest rate, percent.	I	[B]	I
	present value of investment.	PV*	[C]	PV
	periodic payment.	PMT*	[D]	PMT
	future value of investment.	FV*	[E]	FV
5	Compute desired output:			
	number of periods.		[A]	N = (N)
	periodic interest rate.		[B]	I = (I)% (See Note)
	present value of investment.		[C]	PV = $(PV)*
	periodic payment.		[D]	PMT = $(PMT)*
	future value of investment.		[E]	FV = $(FV)*

* Positive for cash received, negative for cash paid out.

Note: Should the routine for *i* fail to return an answer, you may try your own non-zero initial value for *i*. For example, to try a guess of 1%:

.01 [STO] 09 [XEQ] 06

Example 1:

A couple purchases a $50,000 house, borrowing $40,000 at 8.5% for 30 years less one month. What is their monthly payment?

Keystrokes	Display
[a] 40000 [C]	40,000.00
8.5 [ENTER↑] 12 [÷] [B]	0.71
30 [ENTER↑] 12 [×] 1 [−] [A] [D]	PMT=$−307.75

Example 2:

The couple in example 1 sold their house 18 months later, netting $25,000. At what interest rate would they have had to invest their original $10,000 and $307.75 monthly payments to obtain $25,000?

Keystrokes	Display	
18 [A]		
25000 [E]	25,000.00	
10000 [CHS] [C] [B]	I = 3.21%	Monthly interest rate.
12 [×]	38.51	Annual rate.

Program Highlight

This program demonstrates a technique called an interchangeable solution. Each of the five variables in the equation can be written in terms of the remaining four. The five top-row keys are used both for storing inputs and computing outputs using the program structure outlined below.

LBL £	One of the labels A-J or a-e.
STO r	Store the variable in R_r.
FS?C22	Test the digit-entry flag and clear it.
RTN	Stop here if this data was just keyed in.
⋮	Compute the value of the unknown.
STO r	Store the computed value in R_r.
⋮	Display the new value.
RTN	

This building block may be repeated as many times as necessary depending on the number of variables. See lines **16 through 38** for an example of this structure.

Program Listing

```
01♦LBL "FIN"
02 SF 27
03 FIX 2
04♦LBL a
05 CLX
06 STO 01
07 STO 02
08 STO 03
09 STO 04
10 STO 05
11 STO 09
12 "N, I, PV, PMT,F"
13 "⊦V"
14 AVIEW
15 RTN
```

Lines **01 through 15** start the program and initialize the HP-41. Line **02** puts the HP-41 in User mode. Line **03** sets two decimal places. Label a, lines **04 through 15**, stores zero in R_{01} through R_{05} and R_{09}, then puts the key labels in the display.

```
16♦LBL A
17 STO 01
18 FS?C 22
19 RTN
20 RCL 04
21 RCL 09
22 /
23 STO 00
24 RCL 05
25 −
26 RCL 03
27 RCL 00
28 +
29 /
30 LN
31 RCL 09
32 LN1+X
33 /
34 STO 01
35 "N="
36 ARCL X
37 AVIEW
38 RTN
```

Lines **16 through 38** deal with *N*, the number of periods. Line **17** stores the value in the X-register. Line **18** checks the status of flag 22, the numeric data input flag. When set, indicates new data, so stop; else skip line **19**. Lines **20 through 33** calculate new *N*. Line **34** stores the new *N*. Lines **35 through 37** display new *N*.

```
39♦LBL B
40 STO 02
41 1 E2
42 /
43 STO 09
44 1
45 +
46 STO 07
47 RCL 02
48 FS?C 22
49 RTN
```

Lines **39 through 135** deal with *i*, the interest rate. Lines **40 through 46** store *i* and some functions of *i*. Line **48** checks the status of flag 22, the numeric data input flag. When set, indicates new data, so stop; else skip line **49**.

```
50 RCL 04
51 X≠0?
52 GTO 01
53 RCL 05
54 RCL 03
55 /
56 CHS
57 RCL 01
58 1/X
59 Y↑X
60 1
61 −
62 STO 09
63 GTO 00
```

Lines **50** recalls payment. Line **51** checks if payment is zero. If it is, skip line **52** and compute new *i* by simple formula, lines **53 through 63**.

```
64♦LBL 01
65 RCL 05
66 ABS
67 RCL 04
68 RCL 01
69 *
70 RCL 03
71 +
72 ABS
73 −
74 RCL 04
75 RCL 01
76 *
77 RCL 05
78 +
79 ABS
80 RCL 03
81 ABS
82 −
83 *
84 ENTER↑
85 ABS
```

Lines **64 through 126** calculate *i* using Newton's method. Line **86** is the initial guess.

```
86 /
87 1 E−9
88 *
89 STO 09
90♦LBL 06
91 XEQ 08
92 RCL 04
93 *
94 RCL 03
95 +
96 RCL 05
97 RCL 08
98 *
99 +
100 RCL 08
101 RCL 07
102 /
103 RCL 01
104 *
105 STO 06
106 1
107 RCL 08
108 −
109 RCL 09
110 /
111 −
112 RCL 04
113 RCL 09
114 /
115 *
116 RCL 05
117 RCL 06
118 *
119 −
120 /
121 ST− 09
122 ABS
123 1 E−7
124 X<=Y?
125 GTO 06
126 RCL 09
```

Lines **90 through 125** contain the loop that is repeated until Δi is small. This is determined in lines **123 and 124**. The Y-register contains Δi and the X-register contains 1 E−7 (line **123**). Line **124** compares the values in the X- and Y-registers. If $x < y$, then repeat loop; if not, skip line **125**.

```
127♦LBL 00
128 1 E2
129 *
130 STO 02
131 "I="
132 ARCL X
133 "⊦%"
134 AVIEW
135 RTN
```

Lines **127 through 130** store *i*. Lines **131 through 134** display *i*.

```
136♦LBL C
137 STO 03
138 FS?C 22
139 RTN
140 RCL 04
141 XEQ 08
142 *
143 RCL 05
144 RCL 08
145 *
146 +
147 CHS
148 STO 03
149 "PV=$"
150 ARCL X
151 AVIEW
152 RTN
```

Lines **136 through 152** deal with *PV*, present value. Line **137** stores the value in the X-register. Line **138** checks the status of flag 22, the numeric data input flag. When set, indicates new data, so stop; else skip line **139**. Lines **140 through 147** calculate new *PV*. Line **148** stores the new *PV*. Lines **149 through 151** display new *PV*.

```
153♦LBL D
154 STO 04
155 FS?C 22
156 RTN
157 XEQ 08
158 1/X
159 RCL 03
160 RCL 05
161 RCL 08
162 *
163 +
164 *
165 CHS
166 STO 04
167 "PMT=$"
168 ARCL X
169 AVIEW
170 RTN
```

Lines **153 through 170** deal with *PMT*, the payment amount. Line **154** stores the value that is in the X-register. Line **155** checks the status of flag 22, the numeric data input flag. When set, indicates new data, so stop; else skip line **156**. Lines **157 through 165** calculate new *PMT*. Line **166** stores new *PMT*. Lines **167 through 169** display new *PMT*.

```
171♦LBL E
172 STO 05
173 FS?C 22
174 RTN
175 XEQ 08
176 RCL 04
177 *
178 RCL 03
179 +
180 RCL 08
181 /
182 CHS
183 STO 05
184 "FV=$"
185 ARCL X
186 AVIEW
187 RTN
```

Lines **171 through 187** deal with *FV*, the future value. Line **172** stores the value that is in the X-register. Line **173** checks the status of flag 22, the numeric data input flag. When set, indicates new data, so stop; else skip line **174**. Lines **175 through 182** calculate new *FV*. Line **183** stores new *FV*. Lines **184 through 186** display new *FV*.

```
188♦LBL 08
189 1
190 XEQ 09
191 RCL 01
192 CHS
193 Y↑X
194 STO 08
195 −
196 RCL 09
197 /
198 RTN
```

Lines **188 through 198** compute

$$\frac{1-\left(1+\frac{i}{100}\right)^{-n}}{\frac{i}{100}}$$

```
199♦LBL 09
200 RCL 09
201 1
202 +
203 STO 07
204 RTN
```

Lines **199 through 204** compute $1 + i/100$.

Registers Used

R_{00}: Used	R_{05}: FV
R_{01}: N	R_{06}: Used
R_{02}: i	R_{07}: 1 + i/100
R_{03}: PV	R_{08}: Used
R_{04}: PMT	R_{09}: i/100

Flags Used

F22: Digit entry flag	F27: User mode flag

CURVE FITTING

For a set of data points (x_i, y_i), $i = 1, 2, \ldots, n$, this program can be used to fit the data to any of the following curves:

1. Straight line (linear regression): y: $a + bx$.
2. Exponential curve: y: ae^{bx} $(a > 0)$.
3. Logarithmic curve: $y = a + b \ln x$.
4. Power curve: $y = ax^b$ $(a > 0)$.

The regression coefficients a and b are found by solving the following equivalent system of linear equations.

$$An + B\Sigma X_i = \Sigma Y_i$$

$$A\Sigma X_i + B\Sigma X_i^2 = \Sigma Y_i X_i$$

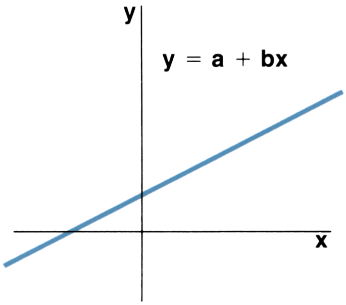
The relations of the variables are defined by the following:

Regression	***A***	***B***	X_i	Y_i
Linear	a	b	x_i	y_i
Exponential	$\ln a$	b	x_i	$\ln y_i$
Logarithmic	a	b	$\ln x_i$	y_i
Power	$\ln a$	b	$\ln x_i$	$\ln y_i$

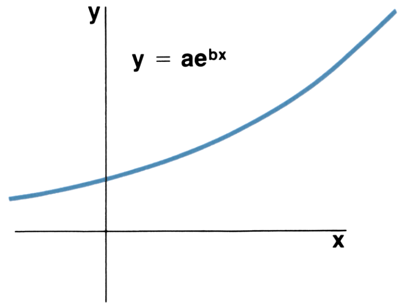
The coefficient of determination is:

$$R^2 = \frac{A\Sigma Y_i + b\Sigma X_i\ Y_i - \frac{1}{n}(\Sigma Y_i)^2}{\Sigma(Y_i^2) - \frac{1}{n}(\Sigma Y_i)^2}$$

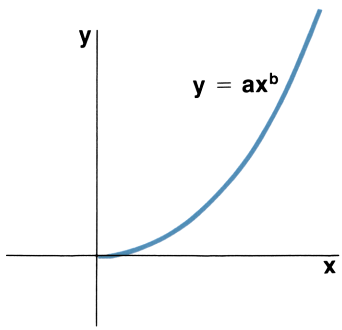
Linear Regression



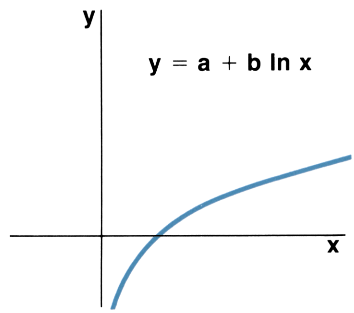
Exponential Curve Fit



Power Curve Fit



Logarithmic Curve Fit



Remarks:

1. The program applies the least square method, either to the original equations (straight line and logarithmic curve) or to the transformed equations (exponential curve and power curve).
2. Negative and zero values of x_i will cause a calculator error for logarithmic curve fits. Negative and zero values of y_i will cause a machine error for exponential curve fits. For power curve fits both x_i and y_i must be positive, non-zero values.
3. As the difference between x and/or y values becomes small, the accuracy of the regression coefficients will decrease.
4. The statistical registers are relocated to R_{10} through R_{15} (line 20).

Running CURVE FITTING

STATUS: SIZE 016, FIX 02

Step	Instructions	Input	Function	Display
1	Set status (above) and key in the program (pages 103–106).			
2	Initialize the program:			
	for STRAIGHT LINE.		LIN	LIN
	or for EXPONENTIAL CURVE.		EXP	EXP
	or for LOGARITHMIC CURVE.		LOG	LOG
	or for POWER CURVE.		POW	POW
3	Repeat step 3 and 4 for $i = 1, 2, \ldots, n$ input:			
	x_i	x_i	ENTER↑	
	y_i	y_i	A	(i)
4	If you made a mistake in inputting x_k and y_k, then correct by:	x_k	ENTER↑	
		y_k	C	($k - 1$)
5	Calculate R^2 and regression coefficients a and b.		E	R2 = (R2)
			R/S	a = (a)
			R/S	b = (b)
6	Calculate estimated y from regression, input x:	x	R/S	Y. = ($\hat{y}$)
7	Repeat step 6 for different x's.			
8	Repeat step 5 if you want the results again.			
9	To use the same program for another set of data, initialize the program by:		a	LIN or EXP or LOG or POW
	then go to step 3.			
10	To use another program, go to step 2.			

Example 1:

Fit a straight line to the following set of data and compute $\hat{y}$ for $x = 37$ and $x = 35$.

x_i	40.5	38.6	37.9	36.2	35.1	34.6
y_i	104.5	102	100	97.5	95.5	94

Keystrokes	Display	
[LIN]	LIN	Remember, to execute a program, press [XEQ] [ALPHA] LIN [ALPHA] or assign the program to a key.
40.5 [ENTER↑] 104.5 [A]	1.00	
38.6 [ENTER↑] 102 [A]	2.00	
37.9 [ENTER↑] 100 [A]	3.00	
36.2 [ENTER↑] 97.5 [A]	4.00	
35.2 [ENTER↑] 95.5 [A]	5.00	Oops!
35.2 [ENTER↑] 95.5 [C]	4.00	Correct error.
35.1 [ENTER↑] 95.5 [A]	5.00	Use proper values.
34.6 [ENTER↑] 94 [A]	6.00	
[E]	R2 = 0.99	
[R/S]	a = 33.53	
[R/S]	b = 1.76	
37 [R/S]	Y. = 98.65	
35 [R/S]	Y. = 95.13	

Example 2:

Fit an exponential curve to the following set of data and compute $\hat{y}$ for $x = 1.5$ and $x = 2$.

x_i	.72	1.31	1.95	2.58	3.14
y_i	2.16	1.61	1.16	.85	0.5

Keystrokes	Display	
EXP	EXP	
.72 ENTER↑ 2.16 A	1.00	
1.31 ENTER↑ 1.61 A	2.00	
1.95 ENTER↑ 1.16 A	3.00	
2.58 ENTER↑ .85 A	4.00	
3.15 ENTER↑ .05 A	5.00	If you don't make a mistake you can skip two steps.
3.15 ENTER↑ .05 C	4.00	
3.14 ENTER↑ 0.5 A	5.00	
E	R2 = 0.98	
R/S	a = 3.45	
R/S	b = −0.58	
1.5 R/S	Y. = 1.44	
2.0 R/S	Y. = 1.08	

Example 3:

Fit a logarithmic curve to the following set of data and compute $\hat{y}$ for $x = 8$ and $x = 14.5$.

x_i	3	4	6	10	12
y_i	1.5	9.3	23.4	45.8	60.1

Keystrokes	Display	
LOG	LOG	
3 ENTER 1.5 A	1.00	
4 ENTER 9.3 A	2.00	
6 ENTER 23.4 A	3.00	
10 ENTER 45.8 A	4.00	
12 ENTER 6.01 A	5.00	Another mistake.
12 ENTER 6.01 C	4.00	
12 ENTER 60.1 A	5.00	
E	R2 = 0.98	
R/S	a = −47.02	
R/S	b = 41.39	
8 R/S	Y. = 39.06	
14.5 R/S	Y. = 63.67	

Example 4:

Fit a power curve to the following set of data and compute $\hat{y}$ for $x = 18$ and $x = 23$.

x_i	10	12	15	17	20	22	25	27	30	32	35
y_i	0.95	1.05	1.25	1.41	1.73	2.00	2.53	2.98	3.85	4.59	6.02

Keystrokes	Display	
[POW]	POW	
10 [ENTER↑] 0.95 [A]	1.00	
12 [ENTER↑] 1.05 [A]	2.00	
15 [ENTER↑] 1.25 [A]	3.00	
17 [ENTER↑] 1.41 [A]	4.00	
20 [ENTER↑] 1.73 [A]	5.00	
22 [ENTER↑] 2.00 [A]	6.00	
25 [ENTER↑] 2.53 [A]	7.00	
27 [ENTER↑] 2.98 [A]	8.00	
30 [ENTER↑] 3.85 [A]	9.00	
32 [ENTER↑] 4.59 [A]	10.00	
35 [ENTER↑] 60.2 [A]	11.00	
35 [ENTER↑] 60.2 [C]	10.00	Error correction.
35 [ENTER↑] 6.02 [A]	11.00	
[E]	R2 = 0.94	
[R/S]	a = 0.03	
[R/S]	b = 1.46	
18 [R/S]	Y. = 1.76	
23 [R/S]	Y. = 2.52	

Program Highlight

This program uses a single section of code for most of the calculations it needs to do. Since each of the four types of curve fitting requires the input data to be in a different form, it would seem that a different program should be used for each curve type. Instead, each of the set-up programs, LIN, LOG, EXP and POW, stores a code in R_{00}. Then the single function on line 32, [XEQ] IND 00, takes care of the four different ways of processing the input data by executing the function whose label is stored in R_{00}.

Program Listing

```
01♦LBL "LIN"
02 5
03 "LIN"
04 GTO 13
```

Lines **01 through 04** begin the linear curve fit program. The 5 in line **02** is the subroutine called indirectly in lines **27** and **32** to process the data.

```
05♦LBL "EXP"
06 6
07 "EXP"
08 GTO 13
```

Lines **05 through 08** begin the exponential curve fit program. The 6 in line **06** is the subroutine called indirectly in lines **27** and **32** to process the data.

```
09♦LBL "LOG"
10 7
11 "LOG"
12 GTO 13
```

Lines **09 through 12** begin the logarithmic curve fit program. The 7 in line **10** is the subroutine called indirectly in lines **27** and **32** to process the data.

```
13♦LBL "POW"
14 8
15 "POW"
```

Lines **13 through 15** begin the power curve fit program. The 8 in line **14** is the subroutine called indirectly in lines **27** and **32** to process the data.

```
16♦LBL 13
17 XEQ "INIT"
18 STO 00
19 ASTO 08
20 ΣREG 10
21 CLΣ
22 BEEP
23 AVIEW
24 STOP
```

Lines **16 through 24** sets up the HP-41 to run the program. Line **17** calls the initialization subroutine. Line **18** stores the subroutine number in R_{00}. Line **19** stores the type of fit in R_{08}. Line **20** sets the statistical registers to begin with R_{10}. Line **21** clears the statistical registers. Line **22** beeps. Line **23** displays the type of fit. Line **24** stops the program, waiting for data input.

```
25♦LBL C
26 X<>Y
27 XEQ IND 00
28 Σ-
29 STOP
```

Lines **25 through 29** contain the subroutine for corrections. Line **27** calls the appropriate data processing routine and line **28** subtracts the x and y values from the statistical registers.

```
30♦LBL A
31 X<>Y
32 XEQ IND 00
33 Σ+
34 STOP
```

Lines **30 through 34** process the data, calling the subroutine stored in R_{00}, then summing the processed data.

```
35♦LBL 07
36 LN
37 RTN
```

Subroutine for logarithmic curve fit.

```
38♦LBL 08
39 LN
```

Subroutine for power curve fit.

```
◆LBL 06
X<>Y
LN
X<>Y
RTN
```

Subroutine for exponential curve fit. Power curve also uses.

```
◆LBL E
RCL 15
RCL 11
RCL 10
RCL 10
XEQ 09
STO 03
RCL 12
RCL 11
RCL 10
RCL 14
XEQ 09
RCL 03
/
STO 04
XEQ IND 00
STO 06
RCL 15
RCL 14
RCL 10
RCL 12
XEQ 09
RCL 03
/
STO 05
◆LBL 03
RCL 04
RCL 12
*
RCL 05
RCL 14
*
+
RCL 12
X↑2
RCL 15
/
STO 09
-
RCL 13
```

Calculate A, b and a, b.

```
85 RCL 09
86 –
87 /
88 "R2"
89 XEQ 88
90 RCL 06
91 "a"
92 XEQ 88
93 RCL 05
94 "b"
95 GTO 01
96♦LBL 06
97♦LBL 08
98 E↑X
99♦LBL 05
100♦LBL 07
101 RTN
```

```
102♦LBL 09
103 *
104 STO 07
105 RDN
106 *
107 RCL 07
108 –
109 RTN
```

Calculates the coefficient of determination.

```
110♦LBL 00
111 "Y."
112♦LBL 01
113 "⊦="
114 ARCL X
115 AVIEW
116 FS? 55
117 STOP
```

Lines **110 through 117** display the value of y.

```
118♦LBL 04
119 GTO IND 00
120♦LBL 08
121 RCL 05
122 Y↑X
123 GTO 09
124♦LBL 06
125 RCL 05
126 *
127 E↑X
```

Input x to calculate $\hat{y}$.

```
128♦LBL 09
129 RCL 06
130 *
131 GTO 00
132♦LBL 07
133 LN
134♦LBL 05
135 RCL 05
136 *
137 RCL 06
138 +
139 GTO 00
```

```
140♦LBL 88
141 "⊦="
142 ARCL X
143 AVIEW
144 RTN
```

Lines **140 through 144** contain the subroutine for displaying results.

```
145♦LBL a
146 GTO IND 08
```

Lines **145 and 146** initializes the program so a new set of data can be entered. Goes to the label stored in R_{08}: LIN, EXP, LOG or POW.

```
147♦LBL "INIT"
148 CLRG
149 FIX 2
150 SF 21
151 SF 27
152 CF 29
153 RTN
```

Lines **147 through 153** initialize the HP-41 by clearing the registers, setting the number of decimal places, setting flags 21 (printer enable flag) and 27 (user mode flag) and clearing flag 29 (to suppress the decimal point).

Registers Used

R_{00}: Index	R_{08}: LIN or EXP or LOG or POW
R_{01}: Not used	R_{09}: $(\Sigma y)^2/n$
R_{02}: Not used	R_{10}: Σx
R_{03}: R^2	R_{11}: Σx^2
R_{04}: A	R_{12}: Σy
R_{05}: b	R_{13}: Σy^2
R_{06}: a	R_{14}: Σxy
R_{07}: Used	R_{15}: n

Flags Used

F21: Printer enable flag	F29: Clear to suppress separator mark
F27: User keyboard flag	F55: Printer existence flag

WORD GUESSING GAME

This program is a version of the word game "hangman." The first player picks a six-character word and gives it to the calculator. The second player guesses various letters until he has completed the word. After each guess, the calculator displays all correctly guessed characters in their appropriate places. When the entire word has been guessed, the number of guesses is displayed.

Running WORD GUESSING GAME

STATUS: SIZE 019, FIX 00

Step	Instructions	Input	Function	Display
1	Set status (above) and key in the program (pages 109–111).			
2	Begin the program.		WORDS	KEY IN WORD
3	First player: Key in your word:	any six characters	R/S	LETTER?
4	Second player: Guess a character:	any character	R/S	word so far LETTER?
5	Repeat step 4 to guess more characters. When word is complete, you will see DONE, WORD is <word>, and YOU TOOK nn GUESSES.			

Example:

Hide "HP41CV" and then guess it.

Keystrokes	Display	
WORDS	KEY IN WORD	
HP41CV R/S	LETTER?	Notice that the program activates the Alpha keyboard.
A R/S		
	LETTER?	

Keystrokes	Display
P [R/S]	P LETTER?
C [R/S]	P C LETTER?
H [R/S]	HP C LETTER?
4 [R/S]	HP4 C LETTER?
1 [R/S]	HP41C LETTER?
V [R/S]	DONE WORD IS<HP41CV> YOU TOOK 7 GUESSES

Program Highlight

Two special routines were used while developing this program: SPEL and DESPEL. SPEL builds up a word from a collection of letters and DESPEL takes apart a word into its component letters. Only DESPEL remains in the program because the job performed by SPEL is done by the letter-comparison portion of the program.

SPEL and DESPEL use indirect addressing to recall or store letters. A loop-control number must be passed to the routines to indicate which registers are to be addressed. The loop-control number must be in the X-register when SPEL or DESPEL is called. The loop-control number is of the form

f1.0ll for SPEL or *ll.0ff* for DESPEL

where

f1 = register for first letter
ll = register for last letter

ff = *f1* − 1

SPEL and DESPEL (or other similar routines) can be used to encode and decode many types of strings.

Program	Comments
01♦LBL "SPEL"	Assumes a cleared Alpha register.
02 STO 07	Store the counter *f1.0ll.*
03♦LBL 08	Build the word.
04 ARCL IND 07	
05 ISG 07	If not last letter, then repeat the loop.
06 GTO 08	
07 RTN	

Program	Comments
01♦LBL "DESPEL"	Store the counter *ll.0ff.*
02 STO 07	
03 ASTO 00	Store the word.
04♦LBL 07	Save all but the last letter.
05 " "	
06 ARCL 00	
07 ASTO 00	
08 ASHF	Save the last letter.
09 ASTO IND 07	
10 DSE 07	If not all letters, then repeat loop.
11 GTO 07	
12 RTN	

Program Listing

```
01♦LBL "WORDS"
02 "KEY IN WORD"
03 AON
04 PROMPT
05 ASTO 08
06 6
07 XEQ "DESPEL"
08 .9
09 STO 17
10 "      "
11 ASTO 09
12 16.01
13 XEQ "DESPEL"
```

Lines **02 through 04** prompt for the secret word. Line **05** stores the word in R_{08}. Line **06** is the counter for DESPEL. Line **07** calls DESPEL which places the letters in the secret word in R_{01} through R_{06}. Line **08 and 09** set up a counter. Lines **10 through 13** put blanks in R_{09} and R_{11} through R_{16}. Line **10** is six spaces.

```
14♦LBL "LTTR"
15 CLA
16 ASTO 09
17 "LETTER?"
18 AON
19 PROMPT
20 ASTO 10
21 ISG 17
22 1.006
23 STO 18
```

Lines **14 through 19** prompt the player for a letter. Line **19** saves the letter in R_{10}. Line **21** adds one to the number of guesses. Lines **22 and 23** initialize the counter in R_{18}.

```
24♦LBL 06
25 " "
26 ASTO Y
27 RCL 18
28 10
29 +
30 CLA
31 ARCL IND X
32 RDN
33 ASTO X
34 X≠Y?
35 GTO 00
36 CLA
37 ARCL 10
38 ASTO Y
39 CLA
40 ARCL IND 18
41 ASTO X
42 X=Y?
43 GTO 00
44 " "
45 ASTO X
```

Lines **24 through 26** store a blank in the Y-register. Line **25** is one space. Lines **27 through 31** recalls the value in the address in the X-register. Line **32**—if position already has letter, then display it. Line **41**—if guess is correct, then display it. Else display blank. Line **44** is one space.

```
46♦LBL 00
47 CLA
48 ARCL 09
49 ARCL X
50 ASTO 09
51 AVIEW
52 10
53 RCL 18
54 +
55 CLA
56 ARCL Y
57 ASTO IND X
58 ISG 18
59 GTO 06
60 CLA
```

Line **48** adds a letter to the display. Line **58** increments the counting loop. Line **65**—if the words are the same, then done, else ask for another guess.

```
61 ARCL 08
62 ASTO Y
63 CLA
64 ARCL 09
65 ASTO X
66 X=Y?
67 GTO 00
68 PSE
69 PSE
70 GTO "LTTR"
```

Lines **74 through 79** display the word. Lines **80 through 87** display the number of guesses.

```
71♦LBL 00
72 "DONE"
73 AVIEW
74 "WORD IS <"
75 ARCL 09
76 "├>"
77 AVIEW
78 PSE
79 PSE
80 RCL 17
81 INT
82 CF 29
83 FIX 0
84 "YOU TOOK "
85 ARCL X
86 "├ GUESSES"
87 AVIEW
88 SF 29
89 FIX 2
90 AOFF
91 RTN
```

Lines **92 through 103** contain the subroutine DESPEL to separate a word into its letters, then store the individual letters indirectly. Line **96** is one blank.

```
92♦LBL "DESPEL"
93 STO 07
94 ASTO 00
95♦LBL 07
96 " "
97 ARCL 00
98 ASTO 00
99 ASHF
100 ASTO IND 07
101 DSE 07
102 GTO 07
103 RTN
```

Registers Used

R_{00}: Temporary	R_{10}: Current letter
R_{01}: First letter, secret word	R_{11}: First letter, player's word
R_{02}: Second letter, secret word	R_{12}: Second letter, player's word
R_{03}: Third letter, secret word	R_{13}: Third letter, player's word
R_{04}: Fourth letter, secret word	R_{14}: Fourth letter, player's word
R_{05}: Fifth letter, secret word	R_{15}: Fifth letter, player's word
R_{06}: Sixth letter, secret word	R_{16}: Sixth letter, player's word
R_{07}: Counter	R_{17}: Counter
R_{08}: Secret word	R_{18}: Counter
R_{09}: Player's word	

Flags Used

F29: Clear to suppress separator mark

BLACKJACK

This program plays a simple version of the card game blackjack (twenty-one). The calculator deals (without replacement) from a 104-card deck, reshuffling when all but 13 cards have been dealt. The player may bet any amount; if he doesn't place a bet, the value of his previous one will be used.

The player and dealer each receive two cards, one of the dealer's cards being exposed. The player may then either draw additional cards (hit) or not draw (stand). The object of the game is to reach, but not exceed, a score of 21 points, counting 10 for face cards, 1 or 11 for aces, and the face value for the remaining cards. If a player's first two cards count 21, he has *blackjack* and immediately collects 1½ times his bet unless the dealer also has blackjack.

When hitting, a player who draws a card bringing his score over 21 is said to *bust* or *be busted* and he loses his bet. When the player stands on a score of 21 or less, the dealer must hit his own hand until his score exceeds 16. At that point the higher hand wins and the player's bank is updated. If the player and dealer should have the same score, the bet is a *stand-off* or a *push*.

Options allowed in casino-style blackjack such as splitting pairs, going down for double, and purchasing insurance are not included in this program.

Running BLACKJACK

STATUS: SIZE 027, FIX 00

Step	Instructions	Input	Function	Display
1	Set status (above) and key in program (pages 116–122).			
2	Assign DL, HT and S to User keys. A seed ($0 \leqslant$ seed $\leqslant 1$) may be placed in R_{00}.			
3	Store your initial bank:	bank	STO 21	
4	To shuffle the deck:		SH	SHUFFLING
5	Place your bet:	BET	DL	I SHOW *c** You have *1* You have *1 2*†
6a	Hit, then repeat this step or go to 6b or		HT	YOU HAVE *cards*
6b	Stand, and the dealer will show his hand and then hit or stand as appropriate.		S	I HAVE *cards* ⋮
7	Repeat from step 5 as desired.			

* *c* is any card, *cards* is a string of cards—the card numbers are linked so a 10 and a 7 will look like 107.

† If you get blackjack in step 5, the display will show BLACKJACK, and [S(TAND)] will be executed automatically.

Example:

Shuffle the deck, key in a seed of .6, and play blackjack using a $2 bet.

Keystrokes	Display	
ASN DL Σ+	ASN DL 11	
ASN HT 1/x	ASN HT 12	
ASN S √x	ASN S 13	
SH	SHUFFLING 104	
0 STO 21		
.6 STO 00		
2 DL	I SHOW 4 YOU HAVE 10K	The DL function is assigned to Σ+.

Keystrokes	Display	
[S]	I HAVE 4J	The [S] function is assigned to [$\sqrt{x}$].
	I HAVE 4J2	
	I HAVE 4J2J	
	BUST	
	YOUR BANK IS $2	
[DL]	I SHOW 2	
	YOU HAVE 92	
[HT]	YOU HAVE 9210	
[S]	I HAVE 24	
	I HAVE 24A	
	YOUR BANK IS $4	

Program Highlights

An interesting portion of this program is the random number generator (lines 07 through 17):

$$r\,n + 1 = \mathrm{FRC}\ (9821 \times r\,n + .211327)$$

This generator was developed by Don Malm as part of an HP-65 Users' Library program. It passes the spectral test (Knuth, *The Art of Computer Programming*, Addison Wesley, Reading, Mass., 1978, V.2, § 3.4) and, because its parameters satisfy Theorem A (op.cit., p. 15), it generates one million distinct random numbers between zero and 1 regardless of the value selected for r_0.

Because the basic random number generator delivers numbers between zero and 1, it is necessary to do further manipulation of the random numbers to get the integers required for the program. By multiplying the random numbers by an integer n, then taking the integer part, numbers from zero to $n-1$ may be generated. This program used the maximum desired number plus 1 to generate numbers from zero to the desired maximum.

With the registers left after keying in this program, you can write a program to play blackjack using simple playing and betting schemes. The routine below check registers and flags used by the blackjack program to determine whether to hit or stand. If the playing program loses, it doubles its bet, eventually winning.

Note that this program requires the memory allocation of atleast 28 data storage registers.

Program	Comment
01♦LBL "PL"	Place new bet
02 2	
03 SF 22	
04♦LBL 02	
05 XEQ "DL"	Deal

06♦LBL 00	
07 RCL 24	Check score
08 12	
09 ENTER↑	Adjustment for ace. If no ace, clear adjustment.
10 10	
11 FS? 07	
12 CLX	
13 −	
14 X<=Y?	If 12 ⩾ score or if blackjack then stand, otherwise hit.
15 GTO 01	
16 FC? 09	
17 GTO 01	
18 XEQ "HT"	
19 GTO 00	
20♦LBL 01	
21 FS? 09	If no blackjack then stand.
22 XEQ "S"	
23 RCL 27	
24 RCL 21	
25 STO 27	Save last bank.
26 −	
27 X<0?	If game won, place new bet.
28 GTO "PL"	
29 X=0?	If game drawn, use last bet.
30 GTO 02	
31 2	If game lost, double the bet.
32 ST* 22	
33 GTO 02	
34 END	

Program Listing

```
01♦LBL "CRD"
02 CLA
03 ASTO 19
04 1
05 STO 15
06 RCL 00
07 9821
08 *
09 .211327
10 +
11 FRC
12 STO 00
13 RCL 14
14 *
15 INT
16 1
17 +
```

Lines **07 through 17** generate the random numbers.

```
18♦LBL 02
19 RCL IND 15
20 X>Y?
21 GTO 03
22 -
23 ISG 15
24♦LBL 99
25 GTO 02
26♦LBL 03
27 DSE IND 15
28♦LBL 99
29 DSE 14
30 12
31 RCL 14
32 X>Y?
33 GTO 04
34 XEQ "SH"
```

Lines **28 through 34** check to see if 12 or less cards remain, and, if true, shuffle the deck.

```
35♦LBL 04
36 RCL 15
37 STO 16
38 10
39 X<=Y?
40 GTO 00
41 X<>Y
42 STO 16
43 1
```

```
44 X=Y?
45 GTO A
46 CLA
47 ARCL Y
48 GTO 01
49♦LBL 00
50 STO 16
51 CLX
52 10
53 X=Y?
54 GTO "10"
55 1
56 +
57 X=Y?
58 GTO J
59 1
60 +
61 X=Y?
62 GTO "Q"
63 "K"
64 GTO 01
65♦LBL A
66 "A"
67 CF 07
68 GTO 01
69♦LBL "Q"
70 "Q"
71 GTO 01
72♦LBL J
73 "J"
74 GTO 01
75♦LBL "10"
76 "10"
77♦LBL 01
78 ASTO 19
79 RCL 16
80 RTN
```

Lines **77 through 80** store the alpha representation of the card.

```
81♦LBL "SH"
82 SF 27
83 CF 29
84 "SHUFFLING"
85 AVIEW
86 1.013
87 ENTER↑
88 8
89♦LBL 14
90 STO IND Y
91 ISG Y
92 GTO 14
93 104
94 STO 14
95 CLD
96 RTN
```

Lines **81 through 96** reconstruct the deck.

```
97♦LBL "DL"
98 CF 09
99 SF 07
100 ABS
101 INT
102 FS?C 22
103 STO 22
104 RCL 22
105 STO 20
106 SF 06
107 CLA
108 ASTO 26
109 ASTO 25
110 XEQ "CRD"
111 RCL 15
112 STO 17
113 XEQ "CRD"
114 STO 23
115 CF 08
116 FS? 07
117 SF 08
118 CLA
119 ARCL 19
120 ARCL 25
121 ASTO 25
122 "I SHOW"
123 ARCL 25
124 AVIEW
125 SF 07
```

Lines **98 and 99** alter flags to indicate blackjack, no ace. **106** specifies whether to use old bet or store a new bet. Line **110** calls the CRD routine to get the dealer's first card. Line **113** calls CRD to get the dealer's second card. Line **117** saves the dealer's A-flag. Line **121** stores the dealer's hand. Lines **122 through 124** display the dealer's up card.

```
126 0
127 STO 24
128 XEQ "CRD"
129 XEQ "PH"
130 XEQ "CRD"
131 XEQ "PH"
132 RCL 24
133 10
134 FS? 07
135 CLX
136 +
137 21
138 X≠Y?
139 SF 09
140 FS? 09
141 RTN
142 21.5
143 STO 24
144 1.5
145 ST* 20
146 "BLACKJACK"
147 AVIEW
```

Line **128** gets the player's first card. Line **130** gets the player's second card. Line **132** displays the player's hand. If no blackjack, line **139** sets flag 9. If blackjack, continue with lines **142 through 147**.

```
148♦LBL "S"
149 CF 06
150 FS? 07
151 GTO 05
152 11
153 RCL 24
154 X>Y?
155 GTO 05
156 10
157 ST+ 24
```

Lines **148 through 157** contain the routine called when player stands.

```
158♦LBL 05
159 CF 07
160 FS? 08
161 SF 07
162 RCL 17
163 STO 15
164 XEQ 04
165 XEQ "DH"
166 FS? 07
167 GTO 07
168 11
169 RCL 23
```

Lines **158 through 167** reinstate dealer's ace-flag, recover dealer's hole card, and display dealer's hand. If no dealer ace, skip to LBL 07.

```
170 X≠Y?
171 GTO 07
172 21.5
173 STO 23
174 "I HAVE BLACKJAC"
175 "⊦K"
176 AVIEW
177 GTO 07
```

Lines **174 through 176** display blackjack when the dealer wins.

```
178♦LBL 06
179 XEQ "CRD"
180 XEQ "DH"
```

Lines **178 through 180** hit the dealer.

```
181♦LBL 07
182 FS? 06
183 GTO 09
184 FC? 09
185 GTO 08
186 RCL 23
187 17
188 X<=Y?
189 GTO 08
190 FS? 07
191 GTO 06
192 11
193 RCL 23
194 X>Y?
195 GTO 06
196 7
197 X>Y?
198 GTO 06
199 10
200 ST+ 23
```

Lines **181 through 200** make playing decisions. If player busted, then settle bets. If player blackjack set the blackjack. If dealer's score is above 17, then settle. If no ace, the dealer hits. Lines **193 through 195**—if ace and score is between 7 and 11, then dealer hits. Lines **199 through 200** add 10 for ace.

```
201♦LBL 08
202 21.5
203 RCL 23
204 X>Y?
205 XEQ "DB"
206 RCL 24
207 -
208 X=0?
209 XEQ "P"
210 X>0?
211 SF 06
```

Lines **201 through 211** check for dealer bust and check for push.

```
212♦LBL 09
213 RCL 20
214 FS? 06
215 CHS
216 ST+ 21
217 "YOUR BANK IS $"
218 ARCL 21
219 AVIEW
220 RTN
```

Lines **212 through 220** adjust and display the bank when the player loses.

```
221♦LBL "HT"
222 XEQ "CRD"
223 XEQ "PH"
224 RCL 24
225 21.5
226 X>Y?
227 RTN
228 "BUST"
229 AVIEW
230 GTO 05
```

Lines **221 through 230** handle a player hit: get a new card, display the new hand, then check for bust.

```
231♦LBL "DB"
232 "BUST"
233 AVIEW
234 0
235 RTN
```

Lines **231 through 235** handle dealer bust.

```
236♦LBL "PH"
237 ST+ 24
238 CLA
239 ARCL 26
240 ARCL 19
241 ASTO 26
242 "YOU HAVE "
243 ARCL 26
244 AVIEW
245 RTN
```

Lines **236 through 245** display the player's hand.

```
246♦LBL "DH"
247 ST+ 23
248 CLA
249 ARCL 25
250 ARCL 19
251 ASTO 25
252 "I HAVE "
253 ARCL 25
254 AVIEW
255 RTN
```

Lines **246 through 255** display the dealer's hand.

```
256♦LBL "P"
257 "A PUSH"
258 AVIEW
259 ST* 20
```

Lines **256 through 259** take care of a push.

Registers Used

R_{00}: Random Number	R_{14}: Number of cards left in deck
R_{01}: Aces	R_{15}: Counter
R_{02}: 2's	R_{16}: Value of current card
R_{03}: 3's	R_{17}: Dealer's hidden card
R_{04}: 4's	R_{18}: Not used
R_{05}: 5's	R_{19}: Value of current card
R_{06}: 6's	R_{20}: Payoff
R_{07}: 7's	R_{21}: Player's bank
R_{08}: 8's	R_{22}: Bet
R_{09}: 9's	R_{23}: Dealer's score
R_{10}: 10's	R_{24}: Player's score
R_{11}: J's	R_{25}: Dealer's hand
R_{12}: Q's	R_{26}: Player's hand
R_{13}: K's	

Flags Used

F06: Player busted	F21: Should match printer existence flag (flag 55)
F07: Set = no Ace; Clear = Ace	F22: Keyboard entry
F08: Set = no dealer Ace; Clear = dealer Ace	F29: Clear to suppress separator mark
F09: Set = no Blackjack; Clear = Blackjack	

Appendices

Appendix A

Error and Status Messages

This appendix lists all error and status messages given by the HP-41CV.

When an illegal operation is attempted on the HP-41, the operation is not performed and an error message appears in the display. To clear the display, press [←]. If the error was caused during a running program, switch to Program mode to see the offending program line.

Some messages are marked as status messages. A status message is for your information and does *not* indicate an error condition.

The variables x, y, and z below refer to the contents of the X-register, the Y-register, and the Z-register, respectively.

Display	Functions	Meaning
ALPHA DATA	Mathematical Any other function using numeric data	Nonnumeric data was used for a function needing numeric data: the X-register (or Y-register, if relevant) contains Alpha data.
DATA ERROR	Mathematical [SDEV]	Illegal math operation with the given operands (division by zero, square root of a negative number).
	[y^x]	$x \leqslant 0$ and $y = 0$, or x is noninteger and $y < 0$.
	[TONE]	$x \geqslant 10$ or $x < 0$.
	[MEAN]	$n = 0$.
	[OCT]	$\lvert x \rvert > 1073741823_{10}$, or x is noninteger.
	[DEC]	x is noninteger, or any digit in x is an 8 or a 9.
	[FACT]	$x < 0$ or is noninteger.
	[%CH]	$y = 0$.
	[FIX] [SCI] [ENG]	$\lvert n \rvert \geqslant 10$.

Display	Functions	Meaning
MEMORY LOST		Continuous Memory has been cleared and reset.
NO	Flags Conditionals	*Status message.* The result of a flag test or conditional test is false.
NONEXISTENT	Storage Recall	One or more registers specified do not exist in data storage.
	ASN GTO XEQ	The label (of a program) specified or called does not exist. (If the function used requires a global label, then specifying a local one also causes this error.)
	ASN XEQ	The function called does not exist. If a catalog-2 function is called, its source device must be attached to the HP-41.
NULL	Any	*Status message.* The function was cancelled by holding its key down.
OUT OF RANGE	Numeric	A number has exceeded the computational or storage capability of the HP-41. Overflow = $\pm 9.999999999\ 99$.
PACKING TRY AGAIN	ASN XEQ GTO . . Program mode	*Status message.* Packing program memory; repeat the operation just attempted. If **TRY AGAIN** appears again, then there is not enough space in main memory to carry out the operation. Try to resize (SIZE).
	SIZE	Packing program memory; repeat the operation. If **TRY AGAIN** repeats, then then there is not enough space to resize.
PRIVATE	Card Reader Custom ROMs	Attempting to view a private program; refer to the owner's handbook for the HP 82104A Card Reader.
RAM	COPY	Attempting to copy into RAM a program whose global label (as specified) is already in RAM (main memory).
ROM		Attempting to alter or access a program that is in ROM (read-only memory, as in an application module).
YES	Flags Conditionals	*Status message.* The result of a flag test or conditional test is true.

Null Characters

Contents

Null Characters and the Alpha Register

The null character is the ‾ (overbar) and corresponds to character code 0.*† Normally the computer does not generate null characters. However, under certain conditions, you can place null characters in Alpha data strings.

Since the null character is not commonly generated, the HP-41 uses the null character as a special indicator. As a result, nulls in the Alpha register occasionally cause unexpected displays, as described in this appendix.

Treatment of Null Characters

The distinction between the Alpha *register* and the Alpha *display* is important when considering the treatment of nulls.

- The Alpha register is always 24 characters long; when it is "empty" it actually contains 24 null characters. As characters enter the Alpha register from the right side, they displace nulls. Any leading nulls (either that you entered or that were already there) remain, but they are ignored by computer operation.
- The Alpha display consists of the characters in the Alpha register *after* the leading nulls. It starts with the first (leftmost) non-null character and displays all others to the right, *including any embedded or trailing nulls.*

The HP-41 and its functions always consider that *an Alpha string starts at the first non-null character*, ignoring leading nulls. Nulls embedded between non-null characters are retained.

---

* The null character has nothing to do with the **NULL** message (which occurs when a function is being cancelled).

† A displayed null is printed as ✱ (which corresponds to character codes 0 and 10) by the HP 82143A and HP 82162A Printers.

Appending Characters. If you append a character to the Alpha register (using ⊦, the append key on the Alpha keyboard), the display will differ from the actual contents of the Alpha register *if the last character (before appending) was a null.*

If the last character in the Alpha register is a null, then—*while you enter characters to append*—the HP-41 acts like the register is empty, and displays only the characters that you are appending. (The input cue (_) is present in the display while you append characters.) *However*, the Alpha register itself properly retains the original string and combines it with the appended string.

You can view the full, appended contents by pressing AVIEW or ALPHA ALPHA. (Remember that leading nulls are never displayed.)

Deleting Characters While Appending. If you use ⊦ or ARCL and the last character in the Alpha string is a null, using ← to delete the rightmost character will *clear the entire Alpha register*. This is because when a null character gets deleted the computer figures that it has encountered the leading nulls that precede a string, and it concludes that the register is empty—so it clears everything.

Alpha Strings in Data or Stack Registers. If you store an Alpha string containing nulls in a data or stack register, none of the nulls will be displayed when you view (or print) the contents of that register (as with VIEW or RCL). However, if you recall those contents to the Alpha register and then view them (ARCL), *all* the characters in the Alpha data string will be displayed (except, of course, leading nulls).

If you print out the Alpha string contents of a data or stack register, only the characters to the left of the first null (the first null from the left) are printed. Any characters to the right of that first null are not printed.

Printer Operation

Contents

Paper Advance

The programmable function ADV (*advance*) causes the printer paper to advance one line. If no printer is attached to the HP-41, ADV has no effect at all. ADV also has no effect if the printer is attached but off, or if flag 21 (below) is clear during a running program.

Controlling Program Execution and Display With Flags 21 and 55

Flag 21 (*printer enable*) and flag 55 (*printer existence*) are set or cleared automatically by the computer each time it is turned on. Normally, then, they are either both cleared or both set: set if a printer is attached, and cleared if no printer is attached.

By using the VIEW or AVIEW functions and manipulating flag 21 (which can be changed by the user; 55 cannot), you can control the display of messages and results during program execution; that is, whether execution stops to show the result or merely displays the result and continues.

The status of flags 21 and 55 determine how VIEW and AVIEW affect a running program. When their status is the same—the usual, default case—operation is normal:

- If no printer is present, VIEW or AVIEW causes the specified register or the Alpha register to be displayed until a later display command places new data in the display. VIEW and AVIEW do not halt program execution.
- If a printer is present *and turned on*, the HP-41 acts as above and, in addition, the displayed data are printed.

There are two reasons to use VIEW and AVIEW in a program. 1) A message can tell you what the program is doing—for example, which subroutine is being executed. However, there is no need for a permanent record of these messages. 2) Other messages give you the results of the program, and you probably want a record of these results. If you don't have a printer, you'll need to halt program execution when results are displayed so you can write them down.

Note that the normal operations above don't halt program execution (to write down data) if a printer is not present, and they record *all* VIEWed data or messages if a printer is present. By clearing or setting flag 21 before executing VIEW or AVIEW, you can control whether the program stops while displaying data and messages regardless of whether a printer is present.

- **Clear flag 21 to display but not record messages.** If flag 21 is clear when VIEW or AVIEW is executed, and no printer is present *or it is off*, the messages and results are displayed and program execution is not halted. This is the first type of normal operation above.

  If a printer is present and turned on, the message is displayed but not printed, and program execution is not halted.

- **Set flag 21 to record results**—whether by printer or by hand. If flag 21 is set when VIEW or AVIEW is executed, and if no printer is present *or the printer is off*, program execution halts so you can write down the displayed result. Press R/S to resume program execution.

  If a printer is present, the result is printed and program execution is not halted. This is the second type of normal operation above.

Therefore, with a printer connected you can still choose whether to print all displays or not. With no printer connected you can choose whether to halt execution or not for displayed results and messages.

Peripherals, Extensions, and HP-IL

Contents

The HP-41 handheld computer becomes a *controller* for a computing system when it is connected to HP peripheral devices and extensions. In addition, the Hewlett-Packard Interface Loop (HP-IL) Module can integrate the HP-41 and up to 30 other devices in a serial communications loop.

Four input/output (I/O) ports are provided on the computer for plugging in system extensions—one device per port. (The HP-IL module uses one port, but each additional HP-IL peripheral does not—it just hooks up by cable to the module or another HP-IL device.)

HP-41 Peripherals

HP 82104A Card Reader

The card reader can record programs, data registers, and key assignments from the HP-41 onto magnetic cards. In turn, programs, registers, and assignments recorded on magnetic cards can then be loaded into the main memory of an HP-41 by the card reader.

The card reader provides quick storage and loading of information (no keying in instructions!). All programs from the Users' Library come with magnetic cards. Furthermore, the card reader can also read cards of HP-67 and HP-97 programs, automatically translating them into the internal code used by the HP-41.

HP 82143A Printer

The printer prints instructions and programs quietly on 24-character-wide thermal paper. The printer can produce upper- and lower-case alphabetic characters, digits, and double-wide characters. There are several printing modes, so you can determine what kinds of output will be printed. This lets you, for instance, check long calculations or diagnose programming problems.

HP 82153A Optical Wand

The wand reads programs encoded in HP bar code, and stores them in the main memory of the HP-41. This is much faster and more accurate than manual key entry; data and individual functions can also be read from bar code into the computer. All Users' Library programs and HP Solutions Books for the HP-41 come with bar code versions of their programs.

Extensions

HP 82182A Time Module

A time module gives your HP-41 a clock, a calendar, a stopwatch, and the ability to set alarms. The alarms can control programs as well as keep appointments for you. In all, this module supplies 29 time-related functions.

HP 82180A Extended Functions/Memory Module

An extended-functions/memory module adds 127 registers of extended memory to your HP-41. Only 124 registers are available to the user because three are used for overhead by the system. These registers can be used to store program files, data files, or text files. The module supplies 47 functions for the creation and modification of these files, as well as for the manipulation of Alpha and numeric data. This module is for people interested in exploiting the programming power of the HP-41 to its fullest.

There is also an HP 82181A Extended Memory Module available, which provides an additional 238 registers of extended memory. You can add one or two of them if you also have an extended-functions module.

Application Pac Modules

The application pac modules are prewritten ROM (read-only memory) software for solving specific problems in specific fields (like Circuit Analysis and Financial Decisions). You can add up to four application pac modules. The programs and functions contained in the application module are listed by catalog 2.

Hewlett-Packard Interface Loop (HP-IL) and Peripherals

By plugging the HP 82160A HP-IL Module into one of the HP-41 ports, you can create a serial interface loop containing up to 30 other HP-IL-compatible devices. The HP-41 itself acts as the controller for the loop, monitoring and controlling the activity of the other devices. The HP-IL module contains the functions necessary to manipulate HP-IL printing and mass storage peripherals.

Among the HP-IL peripherals are devices for mass storage, video display, printing, plotting, and measurement. In addition, the HP 82183A Extended I/O Module extends the function set of the HP-IL module for I/O device control, and the HP 82184A Plotter Module provides advanced plotting capabilities (including bar code formulation). Check with your authorized HP dealer for a complete and up-to-date list of current HP-IL products.

XROM Functions and XROM Numbers

Every user-accessible function or program provided by an HP-41 peripheral or extension is considered an "external ROM" (*XROM*) function. Catalog 2 (below) makes a list of each external device. It can also list every individual function of a source device. Every external ROM function is identified internally by a two-part, XROM number.

Catalog 2: The Catalog of External Functions

Catalog 2 (see also "The Catalogs" in section 1) is a listing of all XROM functions/programs by device. Catalog 2 shows the name of each external source device (the "ROM header") followed by a listing of its individual functions and programs.

[SST] and [BST] work as for other catalogs. When a catalog listing of individual functions reaches the end of the list for that device, the listing goes on to the next source header and the functions for that device.

Programs Versus Functions in External ROM

An operation in ROM in an applications or extension module or in a peripheral is provided either as a *program* or as a *function*. A program can be copied into user memory, then listed and altered, etc. A function, on the other hand, cannot be viewed—only used. When you list out catalog 2, the computer differentiates the two with the "raised T" in front of programs:

```
SECUR 1B  ←—— ROM header (device identification)
ᵀBONDS
ᵀSTOCK    ←—— program
   ⋮
ᵀATP
JDAY      ←—— function
ᵀBEP
   ⋮
```

How XROM Functions Are Displayed as Program Instructions

When an external function is written into a program instruction, the display of that instruction depends on whether or not the module containing that function is currently plugged in to the HP-41, and whether that XROM function is presented as a program or a function.

The XROM number identifies an XROM function by its device (ROM identification number) and its location within that device (function number).

If the necessary module is not plugged in, then the HP-41 has no knowledge of any of its XROM functions—*unless* a function was assigned to a User key, in which case its XROM number is known because it was assigned to that key. Similarly, if a module is removed *after* one of its functions has been entered in a program, the computer identifies the "missing" function by its XROM number. Therefore:

- If the computer currently has access to an XROM function, then it will be entered into a program line as either

label	for an external function, or
XROM^{T}*label*	for an external program.

  This is also the result if a User-defined key is used to enter the program instruction.

- **XROM** *number*, *number* replaces the *label* or **XROM**^{T}*label* display of a program instruction *when the relevant module is removed.* The XROM number remains only as long as its module is missing; that is, the original display is restored when the module is reconnected. This is also the result if a User key is used with the relevant module unplugged.*

* An external function can only be assigned to a User key when the module containing it is connected to the HP-41. Otherwise, the error message **NONEXISTENT** results.

- If the relevant module is not connected and you do an Alpha execution of an XROM function for a program line, then the program line will read simply

  XEQ^T*label*, just like a call for a program in main memory.

  When the module is subsequently restored, the program line does *not* change, and remains

  XEQ^T*label*.

Display of a Program Instruction

A. If the relevant module is plugged in, or a User key is used:

XROM^T*label* (program)

or ——— unplug module ⟶ **XROM** *nn,mm*
⟵ plug module back in ———

label (function)

(This program instruction uses two bytes of memory.)

B. If the relevant module is not plugged in, and a User key is not used:

XEQ^T*label* ——— plug in module ⟶ **XEQ**^{T}*label*
⟵ unplug again ———

(This program instruction uses two bytes plus one byte per character in the label.)

Execution Time. Although the instruction **XEQ**^{T}*label*—entered when the module was out—will work when the module is back in to execute the specified external function/program (case B, above), this instruction is not really equivalent to **XROM**^{T}*label* or *label* (case A, above). Case B is less efficient and will take longer to execute, for the following reason: an XROM call (including simply *label* for an XROM function) goes directly to catalog 2 to search for that particular XROM function. An **XEQ**^{T}*label* command, on the other hand, *first* goes to catalog 1, searching through all the user programs.* When it doesn't find the particular label there, it goes on to catalog 2 to continue the search.

---

* This brings up the interesting point of what happens if you have a user program in main memory with the same global label as the name of an XROM function or program. Since the search for **XEQ**^{T}*label* always starts with catalog 1, it will always execute the user program, and not the XROM function. This feature allows you to copy a program from an external ROM module into main memory, modify it, and then execute the modified version rather than the ROM module version even when the module is plugged in.

Memory Space. An **XROM**$^{\mathsf{T}}$*label* or *label* instruction (case A) requires two bytes of memory, while an **XEQ**$^{\mathsf{T}}$*label* instruction requires two bytes plus one byte per character in the label.

Duplicate XROM Numbers

All plug-in ROM modules have ROM identification numbers, and some of them are duplicated. *Avoid simultaneously using any ROM modules with duplicate ROM identification numbers.* (Internal functions do not have XROM numbers.)

Function Tables

Function Tables

Contents

Introduction

These tables describe the functions in the computer. Each table describes functions with common characteristics, and some functions appear in more than one table. Most tables include the information found in "Explanation of Table Entries."

Locating a Function

- To find a function that performs a particular operation, look through the function table whose title describes the desired type of operation.
- To find out what a function does when you know only its name, refer to the Function Index inside the back cover. The last page reference listed will direct you to the proper function table.

Explanation of Table Entries

Alpha Name. This is how the the function is named in catalog 2 or 3, in a program listing, and when you hold down a key for function preview. This is how you must specify the function to assign it to the User keyboard; if the function has no entry in this column, you can't assign it to the User keyboard.

Keyboard Name. This is how the function is indicated on the Normal or Alpha keyboard. (If the entry is printed in gold, you must press ■ before the appropriate key.) If the function has no entry in this column, you must use [XEQ] and the Alpha name or else assign the function to the User keyboard.

IND. An "I" in this column indicates that you can indirectly specify the parameter for this function. To do so, enter the function and press ■; **IND** will then appear in the display following the function name. Then specify the register holding the *address* of the register to access.

Stack. This shows how the function affects the automatic memory stack.

L = LAST X. The previous contents of the X-register are copied into the LAST X register.
↓ = The stack drops. The contents of the Z-register are copied into the Y-register and the contents of the T-register are copied into the Z-register.
↑ = The stack lifts. The contents of the X-, Y- and Z-registers are copied into the Y-, Z-, and T-registers respectively; the previous contents of the T-register are lost. (This assumes that stack lift was previously enabled.)
E = Stack lift enabled. If the next function executed shows "↑" in the "Stack" column or if you key in a number, the stack will lift. (Almost all functions enable stack lift.)
D = Stack lift disabled. If the next function executed shows "↑" in the "Stack" column or if you key in a number, the new number in the X-register replaces the previous contents and the stack doesn't lift. (Only [CLx], [ENTER↑], [Σ+], and [Σ−] disable stack lift.)
N = Neutral. Stack lift is neither enabled nor disabled; the previous status is maintained.

Flags. These are the flags that affect or are affected by the function's operation.

Bytes. This is the number of bytes of program memory required when the function is used in a program. If the function has no entry in this column, it is not programmable.

System/Format Functions

Most of these functions involve options that remain in effect indefinitely: display formats, angular mode, main memory allocation, User-keyboard assignments, and so on. Included are certain system operations such as the toggle keys and the catalogs.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
	ALPHA	Activates/deactivates Alpha keyboard.			48		9
AOFF		Deactivates Alpha keyboard.		E	48	1	13
AON		Activates Alpha keyboard.		E	48	1	13
ASN	ASN	Assigns specified function or global label to specified key on User keyboard.		E			20
CAT n	CATALOG n	Executes catalog n, $1 \leqslant n \leqslant 3$.					
		Catalogs 1, 2, 3.		N			23
CF nn	CF nn	Clears flag nn, $00 \leqslant nn \leqslant 29$.	I	E	nn	2	62
DEG		Selects decimal Degrees angular mode.		E	42-43	1	38
ENG n	ENG n	Selects engineering display format with $n + 1$ digits.	I	E	36-41	2	15
FIX n	FIX n	Selects fixed-point display format with n decimal places.	I	E	36-41	2	14
GRAD		Selects Grads angular mode.		E	42-43	1	38
	ON	Turns computer on/off.			11-26, 45-55		9
ON		Selects continuous on (disables time-out).			44		9
	PRGM	Enters/exits Program mode.		N			9
RAD		Selects Radians angular mode.		E	42-43	1	38
SCI n	SCI n	Selects scientific display format with n decimal places.	I	E	36-41	2	14
SF nn	SF nn	Sets flag nn, $00 \leqslant nn \leqslant 29$.	I	E	nn	2	62
ΣREG nn		Assigns statistics registers to R_{nn} through R_{nn+5}.	I	E		2	42

System/Format Functions (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
SIZE *nnn*		Allocates *nnn* main memory registers for data storage.		E			50
	USER	Activates/deactivates User keyboard.		N	27		9

Clearing Functions

To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Descripcription	Stack	Flags	Bytes	Page
	←	When input cue (_) is displayed, clears last digit or character entered. When digit or character entry is terminated, clears X-register or Alpha register in Execution mode; deletes displayed program line in Program mode.	*			12 13 59
		When message is displayed, clears message.	N	50		
	← down, ON, ← up	Clears all of computer's memory except for clock time and date.		00-55		9
CLA	CLA	Clears Alpha register.	E		1	13
CLD		Clears message from display.	E	50	1	77
CLP *label*		Clears the program in main memory containing specified global label.	E			60
CLRG		Clears all data storage registers in main memory.	E		1	53
CLΣ	CLΣ	Clears statistics registers.	E		1	42
CLST		Clears automatic memory stack.	E		1	35
CLX	CLx	Clears X-register.	D		1	12
DEL *nnn*		Deletes *nnn* program lines, starting with displayed line.	N			60

* When pressing ← clears the X-register, stack lift is disabled. Otherwise, ← is neutral.

Stack/Data Register Functions

These functions manipulate the stack or the data storage registers, or take one of those registers as a parameter. To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ARCL *nn*	ARCL *nn*	Appends contents of R_{nn} to Alpha register.	I	E	28, 29, 36-41	2	51
ASTO *nn*	ASTO *nn*	Copies six leftmost characters in Alpha register into R_{nn}.	I	E		2	52
CLRG		Clears all data storage registers.		E		1	52
CLΣ	CLΣ	Clears statistics registers		E		1	42
CLST		Clears automatic memory stack.		E		1	35
CLX	CLx	Clears X-register.		D		1	12
DSE *nn*		For *iiiii.fffcc* in R_{nn}, decrements *iiiii* by *cc* and skips next program line if $iiiii - cc \leqslant fff$.	I	E		2	75
ENTER↑	ENTER↑	Copies number in X-register into Y-register and lifts stack.		↑, D		1	27
ISG *nn*	ISG *nn*	For *iiiii.fffcc* in R_{nn}, increments *iiiii* by *cc* and skips next program step if $iiiii + cc > fff$.	I	E		2	75
LASTX	LASTx	Recalls number in LAST X register.		↑, E		1	31
R↑		Rolls up stack.		E		1	23
RCL *nn*	RCL *nn*	Recalls contents of R_{nn}.	I	↑, E		*	51
RDN	R↓	Rolls down stack.		E		1	33
Σ+	Σ+	Accumulations for statistics.		L,D		1	43
Σ−	Σ−	Corrects statistics accumulations.		L,D		1	43

* If $00 \leqslant nn \leqslant 15$, requires 1 byte; otherwise, requires 2 bytes.

Stack/Data Register Functions (coninued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ΣREG *nn*		Assigns statistics registers to R_{nn} through R_{nn+5}.	I	E		2	42
SIZE *nnn*		Allocates *nnn* main memory registers for data storage.		E			50
ST+ *nn*	STO + *nn*	Adds number in X-register to number in R_{nn} and places result in R_{nn}.	I	E		2	52
ST− *nn*	STO − *nn*	Subtracts number in X-register from number in R_{nn} and places result in R_{nn}.	I	E		2	52
ST* *nn*	STO × *nn*	Multiplies number in X-register by number in R_{nn} and places result in R_{nn}.	I	E		2	52
ST/ *nn*	STO ÷ *nn*	Divides number in X-register into number in R_{nn} and places result in R_{nn}.	I	E		2	52
STO *nn*	STO *nn*	Copies contents of X-register into R_{nn}.	I	E		*	51
VIEW *nn*	VIEW *nn*	Displays contents of R_{nn}.	I	E	21,50, 55	2	77
X<> *nn*		Exchanges contents of X-register with contents of R_{nn}.	I	E		2	52
X<>Y	x≷y	Exchanges contents of X-register with contents of Y-register.		E		1	33

* If 00 ≤ *nn* ≤ 15, requires 1 byte; otherwise, requires 2 bytes.

Numeric Functions

All numeric functions are programmable, requiring one byte of program memory. The operation of trigonometric functions and rectangular/polar coordinate conversions depends on the angular mode (flags 42 and 43). To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Description	Stack	Page
+	+	$y + x$.	L,↓,E	40
−	−	$y - x$.	L,↓,E	40
*	×	$y \times x$.	L,↓,E	40
/	÷	y / x.	L,↓,E	40
1/X	1/x	Reciprocal.	L,E	37
10↑X	10^x	Common exponential.	L,E	39
ABS		$\lvert x \rvert$ (Absolute value).	L,E	38
ACOS	COS^{-1}	Arc (inverse) cosine.	L,E	38
ASIN	SIN^{-1}	Arc (inverse) sine.	L,E	38
ATAN	TAN^{-1}	Arc (inverse) tangent.	L,E	38
CHS	CHS	Change sign.	E	37
COS	COS	Cosine.	L,E	38
D−R		Degrees to radians conversion.	L,E	39
DEC		Octal to decimal conversion.	L,E	39
E↑X	e^x	Natural exponential.	L,E	39
E↑X−1		Natural exponential for arguments close to zero.	L,E	39
FACT		$x!$ (Factorial).	L,E	37
FRC		Fractional part.	L,E	38
HMS		Decimal hours to hours-minutes-seconds conversion.	L,E	39
HMS+		Hours-minutes-seconds add.	L,↓,E	40
HMS−		Hours-minutes-seconds subtract.	L,↓,E	40
HR		Hours-minutes-seconds to decimal hours conversion.	L,E	39
INT		Integer part.	L,E	38
LN	LN	Natural logarithm.	L,E	39
LN1+X		Natural logarithm for arguments close to 1.	L,E	39

Numeric Functions (continued)

Alpha Name	Keyboard Name	Description	Stack	Page
LOG	LOG	Common logarithm.	L,E	39
MEAN		Means of accumulated *x*- and *y*-values.	L,E	44
MOD		*y* mod *x* (Remainder).	L,↓,E	42
OCT		Decimal to octal conversion.	L,E	39
P–R	P→R	Polar to rectangular conversion.	L,E	41
%	%	*x* percent of *y*.	L,E	40
%CH		Percent change from *y* to *x*.	L,E	40
PI	π	Pi (3.141592654).	↑,E	13
R–D		Radians to degrees conversion.	L,E	39
R–P	R→P	Rectangular to polar conversion.	L,E	41
RND		Round.	L,E	38
SDEV		Standard deviations of accumulated *x*- and *y*-values.	L,E	44
Σ+	Σ+	Accumulations for statistics.	L,D	43
Σ–	Σ–	Accumulations correction.	L,D	43
SIN	SIN	Sine.	L,E	38
SIGN		Sign of *x*.	L,E	38
SQRT	$\sqrt{x}$	Square root.	L,E	37
TAN	TAN	Tangent.	L,E	38
X↑2	x^2	Square.	L,E	37
Y↑X	y^x	*y* raised to the *x* power.	L,↓,E	41

Editing Functions

These are non-programmable functions that are executed in Program mode. They help you write or edit your programs. Like the toggle keys [ON], [USER], and [ALPHA], these functions don't require you to return to Execution mode for execution. To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Description	Flags	Page
	[←]	When input cue (_) is displayed, clears last digit or character entered; otherwise, clears displayed program line.		59
[ASN]	[ASN]	Assigns specified function or global label to specified key on User keyboard.		20
[BST]	[BST]	Displays preceding program line.		58
[CAT] *n*	[CATALOG] *n*	Executes catalog *n*, $1 \leqslant n \leqslant 3$.		23
[CLP] *label*		Clears program in main memory containing specified global label.		60
[COPY] *label*		Copies ROM program containing specified global label to program memory.		55
[DEL] *nnn*		Deletes *nnn* program lines, starting with displayed line.		60
	[GTO] [·]	Goes to specified line number or global label.		57
	[GTO] [·] [·]	Goes to bottom of program memory; packs program memory and creates null program.		55
[ON]		Selects continuous on (disables time-out).	44	9
[PACK]		Packs program memory.		50
[SIZE] *nnn*		Allocates *nnn* main memory registers for data storage.		50
[SST]	[SST]	Displays next program line.		58

Functions That Direct Program Execution

These are functions that can halt program execution or cause program lines to be executed other than sequentially. To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
AVIEW	AVIEW	Displays contents of Alpha register; if flag 21 is set and flag 55 is clear, stops program execution.		E	21, 50, 55	1	77
DSE *nn*		For *iiiii.fffcc* in R_{nn}, decrements *iiiii* by *cc* and skips next program line if $iiiii - cc \leqslant fff$.	I	E		2	75
END		Marks end of program.		E		3	71
FC? *nn*		Tests flag *nn* ($00 \leqslant nn \leqslant 55$) and skips next program line unless flag *nn* is clear.	I	E	*nn*	2	74
FC?C *nn*		Tests flag *nn* ($00 \leqslant nn \leqslant 29$), clears flag *nn*, and then skips next program line unless flag *nn* was clear.	I	E	*nn*	2	74
FS? *nn*	FS? *nn*	Tests flag *nn* ($00 \leqslant nn \leqslant 55$) and skips next program line unless flag *nn* is set.	I	E	*nn*	2	74
FS?C *nn*		Tests flag *nn* ($00 \leqslant nn \leqslant 29$), clears flag *nn*, and then skips next program line unless flag was set.	I	E	*nn*	2	74
GTO *label*	GTO *label*	Transfers execution to specified global, numeric, or local Alpha label.	I	E		*	70
ISG *nn*	ISG *nn*	For *iiiii.fffcc* in R_{nn}, increments *iiiii* by *cc* and skips next program step if $iiiii + cc > fff$.	I	E		2	75

* If $00 \leqslant nn \leqslant 14$ or parameter is indirectly specified, requires 2 bytes; if parameter is global label of *m* characters, requires $2 + m$ bytes; otherwise, requires 3 bytes.

Functions That Direct Program Execution (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
LBL	LBL	Global, numeric, or local Alpha label.		E		*	69
OFF		Turns off the computer.		N	11-26 44-55	1	65
PROMPT		Displays contents of the Alpha register and stops execution.		E	50	1	77
RTN	RTN	Returns execution to line following XEQ instruction that called this subroutine.		E		1	71
STOP	R/S	Stops execution.		E		1	72
VIEW *nn*	VIEW *nn*	Displays contents of R_{nn} and, if flag 21 is set and flag 55 is clear, stops execution.	I	E	21, 50, 55	2	77
X = 0?	x = 0?	Skips next instruction unless number in X-register = 0.		E		1	74
X ≠ 0?		Skips next instruction unless number in X-register ≠ 0.		E		1	74
X < 0?		Skips next instruction unless number in X-register < 0.		E		1	74
X <= 0?		Skips next instruction unless number in X-register ≤ 0.		E		1	74
X > 0?		Skips next instruction unless number in X-register > 0.		E		1	74
X = Y?	x = y?	Skips next instruction unless contents of X-register = contents of Y-register.		E		1	74
X ≠ Y?		Skips next instruction unless contents of X-register ≠ contents of Y-register.		E		1	74
X < Y?		Skips next instruction unless number in X-register < number in Y-register.		E		1	74
X <= Y?	x ≤ y?	Skips next instruction unless number in X-register ≤ number in Y-register.		E		1	74

* If 00 ≤ *nn* ≤ 14, requires 1 byte; if parameter is global label of *m* characters, requires 4 + *m* bytes; otherwise, requires 2 bytes.

Functions That Direct Program Execution (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
X > Y?	x > y?	Skips next instruction unless number in X-register > number in Y-register.		E		1	74
XEQ *label*	XEQ *label*	Calls specified global, numeric, or local Alpha label as subroutine.	I	E		*	71
		(If you specify a function, refer to the table entry for that function.)					71

* If label is specified indirectly, requires 2 bytes; if local label is specified, requires 3 bytes; if global label of *m* characters is specified, requires 2 + *m* bytes.

Alpha Functions

These functions involve moving data into and out of the Alpha register, and manipulating the data in the Alpha register. Not included are functions that use the Alpha register for a file name. To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes
	⊦	Appends subsequent characters to Alpha register.		N		
AOFF		Deactivates Alpha keyboard.		E	48	1
AON		Activates Alpha keyboard.		E	48	1
ARCL *nn*	ARCL *nn*	Appends contents of R_{nn} to Alpha register.	I	E	28, 29, 36-41	2
ASHF		Shifts six leftmost characters out of the Alpha register.		E		1
ASTO *nn*	ASTO *nn*	Copies six leftmost characters in Alpha register into R_{nn}.	I	E		2
AVIEW	AVIEW	Displays contents of Alpha register.		E	21, 50, 55	1
CLA	CLA	Clears Alpha register.		E		1
PROMPT		Displays contents of Alpha register and stops program execution.		E	21, 50, 55	1

Interactive Functions

To interpret this table, refer to "Explanation of Table Entries" on page 138.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ADV		Advances paper (if printer is present).		E	21,55	1	84
BEEP	BEEP	Sounds four tones.		E	26	1	78
PROMPT		Displays contents of Alpha register and stops execution.		E	50	1	77
PSE		Delays execution for about one second.		E		1	77
TONE *n*		Sounds tone n, $0 \leqslant n \leqslant 9$.	I	E	26	2	78

Indexes

Subject Index

Page numbers in **bold** type indicate primary references; page numbers in regular type indicate secondary references.

R

S

Function Index

For each function, its Alpha name is given first (in blue), and its keyboard name follows (in black or gold), although not all functions have both an Alpha name and a keyboard name. (These conventions are explained on the inside of the front cover.)

Each function has up to two page references. The first one is for the text, while the second one, in **boldface**, is for the Function Tables.

HEWLETT PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

00041-90531 Rev. B English

Printed in U.S.A. 4/86