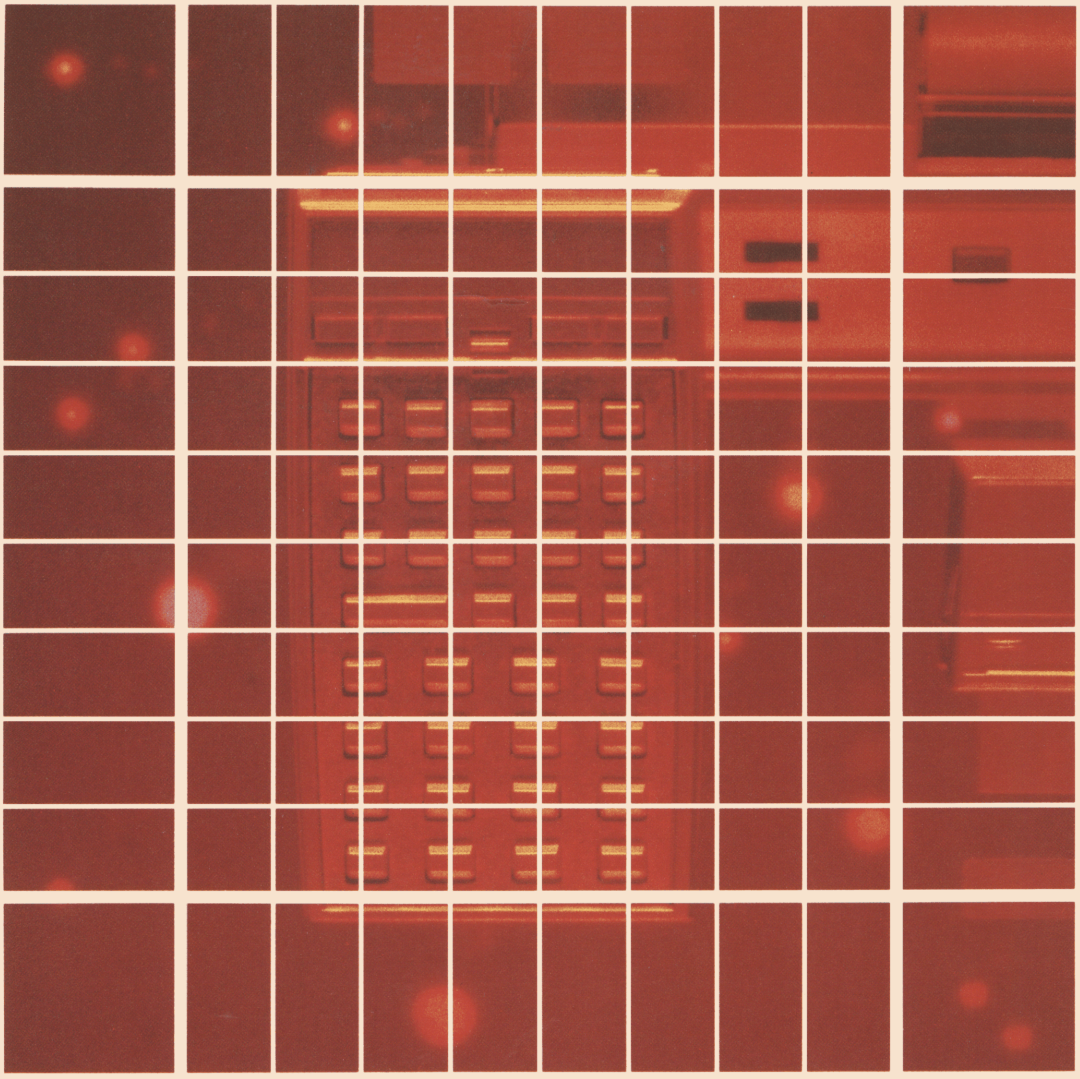
HEWLETT-PACKARD

HP-41CX

OWNER'S MANUAL

VOLUME 2: 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.
[DATE]	*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.
[T] [T] [.] [T]	*Black letters in keyboxes.* These are special functions, *not* Alpha characters, and are active only in special circumstances. If it is a shifted function, it is preceded by
parameter	*The type of parameter* required for a function.

For a full description, refer to "How This Manual Represents Keystrokes," page 16.

HEWLETT PACKARD

HP-41CX
Owner's Manual

Volume 2
Operation in Detail

August 1983

00041-90492

Printed in Singapore

Introducing Volume 2

This is the second volume of the two-volume *HP-41CX Owner's Manual.* Volume 1, *Basic Operation,* is an extensive introduction to most aspects of the HP-41CX. This second volume, *Operation in Detail,* is an advanced, detailed examination of *all* aspects of the HP-41CX. Together, these two volumes form one manual, so the page numbers in this volume continue sequentially from where those in volume 1 left off. The numbers of the sections and the parts also continue in sequence from volume 1.

"How To Use This Manual" on page 9 (volume 1) explains the scheme of this handbook and recommends places to start reading. Look there for a brief overview of the two volumes. This volume (volume 2) emphasizes *completeness* of information and *reference* information. All the appendices are in this volume, as is an index to all the functions (inside the back cover) and a comprehensive summary of all the functions in the Function Tables (the blue-edged pages in front of the Subject Index). If you are already familiar with the HP-41, remember to check appendix I, "A Comparison With the HP-41C/CV."

Volume 2: Operation in Detail

Contents

Part II: Fundamentals in Detail

Part III: Memory in Detail

Part IV: Time Functions in Detail

Part V: Programming in Detail

Appendices

List of Diagrams and Tables

Diagrams

Tables

Contents of Volume 1: Basic Operation

Part II:
Fundamentals in Detail

Section 9

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 and extended memory and the status of certain flags. To reset the computer (that is, to clear main and extended 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 [f] 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 [f] 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 [f] 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.

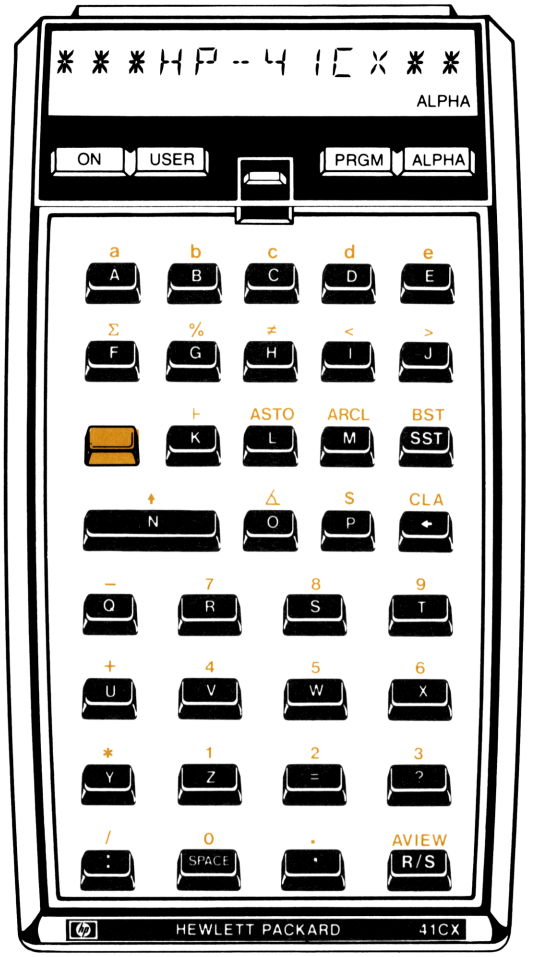
Because shifted key positions can be redefined as well, one key can execute four different functions, depending on whether the User keyboard is active and whether the [f] shift key is pressed first. The operation of the User keyboard is described on page 166.

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.

Special Keyboards

In addition to the keyboards activated by the [USER] and [ALPHA] toggle keys, there are special keyboards activated by functions. Some major examples are:

- The Text Editor keyboard, activated by [ED]. Based on the Alpha keyboard, this keyboard includes commands to manipulate a text file in extended memory. The [ED] function is described in section 14.
- The Alarm Catalog keyboard, activated by [ALMCAT] and [CATALOG] 5. These functions enable you to examine and alter alarms in memory. Alarms are described in section 16.
- The Stopwatch keyboard, activated by [SW] and [SWPT]. Stopwatch operations are described in section 17.

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.*

* 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.

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, [f], 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].

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.

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 G.
- **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.

Some status annunciators have special meanings when [ED] is operating. They return to their previous states when you exit [ED].

Numeric Display Format

The computer represents every number internally in the form $a \times 10^b$ where a is number with nine decimal places, $1 \leq |a| < 10$, and b is a two-digit integer, $0 \leq |b| < 100$. 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] on page 186.) 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 ($0 \leq n \leq 9$). 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 ($0 \leq n \leq 9$), multiplied by a power of 10. For $n \leq 7$, 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. Examples include the displays for the text editor (section 14), the clock (section 15), the stopwatch (section 17), and a program's messages for the user (section 21). 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

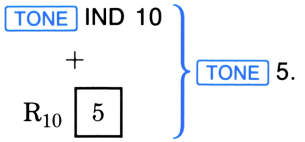
The parameters for most functions can be specified indirectly: rather than entering the parameter itself in response to the input cue, you enter the address of a register (the "indirect register") that contains the parameter. This feature is particularly useful when the value of the parameter depends 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:

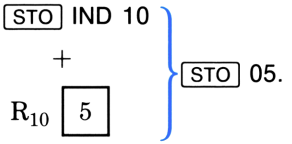
1. Execute the function.
2. In response to the input cue, press [ ]. The display will show **IND** _ _ after the function name.
3. Specify the indirect register.

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.

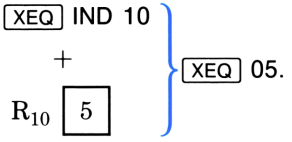


Example. Suppose that R_{10} contains 5. If you execute [STO] IND 10, the address in R_{10} becomes the parameter for [STO]. Therefore, [STO] IND 10 is equivalent to [STO] 05 when R_{10} contains 5.



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 [RCL] IND 10, [1/x], [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 20, "Branching".)

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



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.

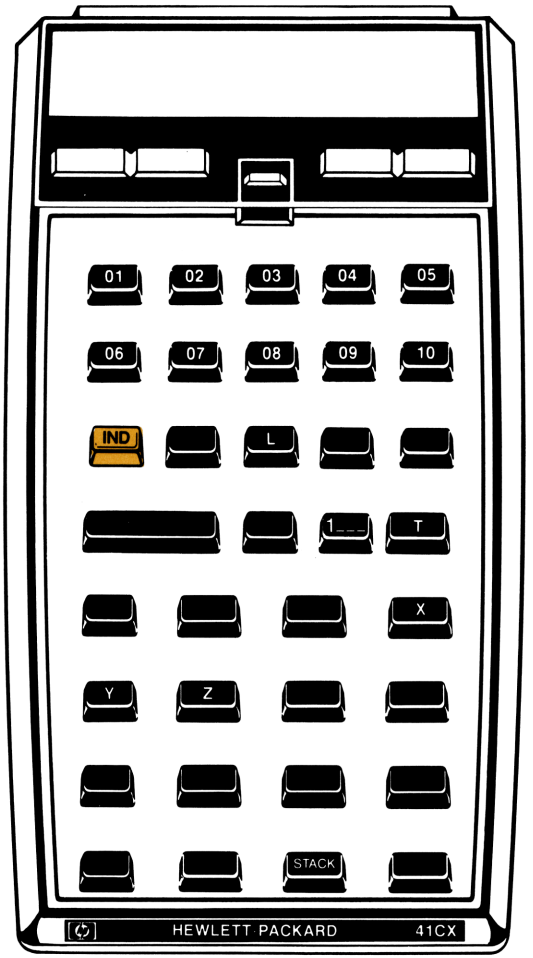
  [STO], [RCL].
  [STO] [+], [STO] [−], [STO] [×], [STO] [÷].
  [ASTO], [ARCL].
  [ISG], [DSE].
  [X<>], [VIEW], [ΣREG].

- [XEQ], [GTO].
- [SF], [CF], [FS?], [FC?], [FS?C], [FC?C].
- [FIX], [SCI], [ENG].
- [TONE].

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 are two functions that assign functions and global labels to the User keyboard, [ASN] (*assign*) and [PASN] (*programmable assign*). Use [ASN] to make assignments manually; it's easier to use but isn't programmable. Use [PASN] to make assignments under program control.

To make an assignment manually:

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

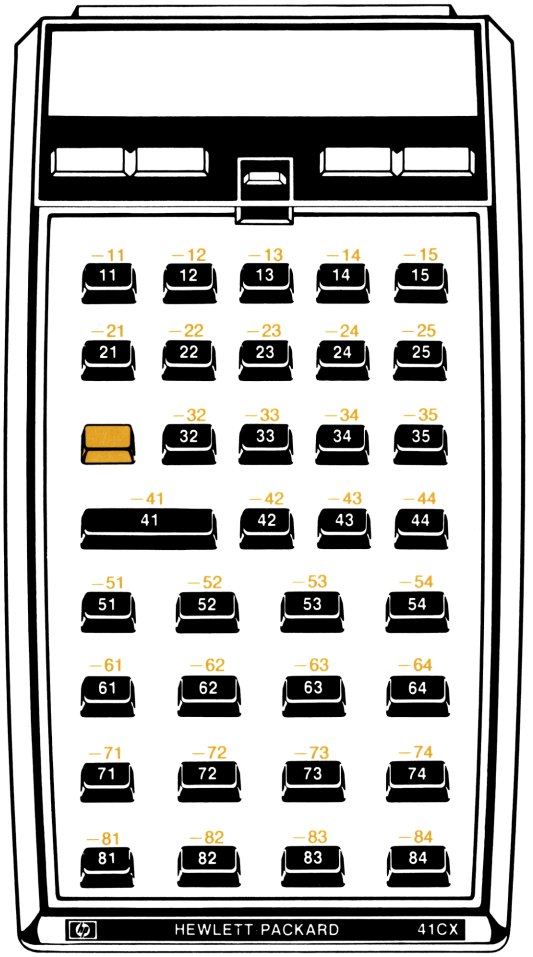
To make an assignment under program control:

1. Enter the function name or global label into the Alpha register.
2. Enter the key code of the key to be redefined (according to the diagram on the facing page) into the X-register.
3. Execute [PASN].

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.

The User Keyboard



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 12.) 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 User Keyboard Catalog

Executing [CATALOG] 6 displays all functions and global labels assigned to the User keyboard. The order of display is by keycode, with smaller numbers before larger numbers (left to right and top to bottom on the keyboard) and positive before negative (unshifted before shifted). You can stop the listing by pressing [R/S], and then:

- Press [SST] to see the next assignment.
- Press [BST] to see the previous assignment.
- Press [f] [C] to clear the displayed assignment.
- Press [R/S] to restart the listing.

If there are no assignments or if the only assignment is cleared using [f] [C], the computer displays **CAT EMPTY**

Restoring Normal Functions

You can cancel the assignment to a redefined key by any one of the following methods:

- Executing [ASN], pressing [ALPHA] twice, and then pressing the appropriate key.
- Clearing the Alpha register, entering the appropriate keycode into the X-register, and executing [PASN].
- Pressing [f] [C] when catalog 6 is halted and displays the assignment.

To cancel all assignments currently in effect, execute [CLKEYS]. Note that assignments that are stored in global labels in extended memory will be reactivated if the User keyboard is active (flag 27 is set) when the program is recalled.

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 20, "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.

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 six catalogs that enable you to review memory contents. The [CATALOG] function is not programmable, but there are programmable functions equivalent to catalogs 4 and 5. 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] speeds up 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.

Catalogs 4, 5, and 6 consume as much power as a running program even when the listing is stopped. Therefore, the computer exits these catalogs after about two minutes of inactivity.

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 each [END] instruction appears the number of bytes in that program; 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. (Page 284.)

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. Initially the catalog lists only the main entries (headers) in each group. To list all entries, press [R/S] to stop the listing, wait for the display to blink, and then press [ENTER↑]. To return to a listing of headers only, press [R/S] to stop the listing and then press [ENTER↑]. (Page 394.)

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.

Catalog 4: Extended Memory Directory. A list of all files in extended memory. The name, type, and number of registers for each file is shown. After listing all existing files, the computer displays the number of registers available for a new file. A program can execute this catalog as [EMDIR].

You can use catalog 4 to make any file the current file: press [R/S] to stop the listing at the desired file, and then press [←] to exit the catalog. (Page 206.)

Catalog 5: Alarm Catalog. A list of all alarms in alarm memory. The time, date, and message for each alarm is shown. You can delete alarms, reset repeating alarms, and look at specific parts of the alarm using the Alarm Catalog keyboard. A program can execute this catalog as [ALMCAT]. (Page 255.)

Catalog 6: User Keyboard Catalog. A list of all functions and global labels assigned to the User keyboard. The name of the function or global label and the key code indicating key location is shown for each assignment, starting at keycode 11 ([Σ+]) and ending at −84 ([VIEW]).

You can use catalog 6 to cancel any assignment: press [R/S] to stop the listing at the desired assignment, and then press [■] [C]. (Page 168.)

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).

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.

* Flags 24 and 25 can prevent certain anticipated errors from stopping program execution. These flags are described on page 290.

Section 10

The Automatic Memory Stack

Contents

Introduction

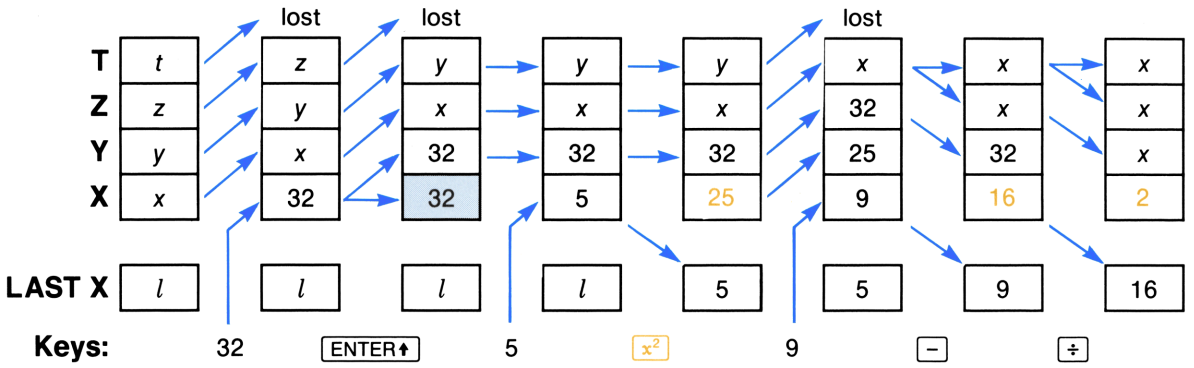
Numeric functions use four registers called the *automatic memory stack.* Numbers automatically move "up and down" in the stack when you enter numbers and perform calculations. The logic used is Reverse Polish Notation (RPN), which minimizes keystrokes and produces all intermediate results. If you are unfamiliar with RPN, refer to page 20.

- 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, 3, and 6.

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.

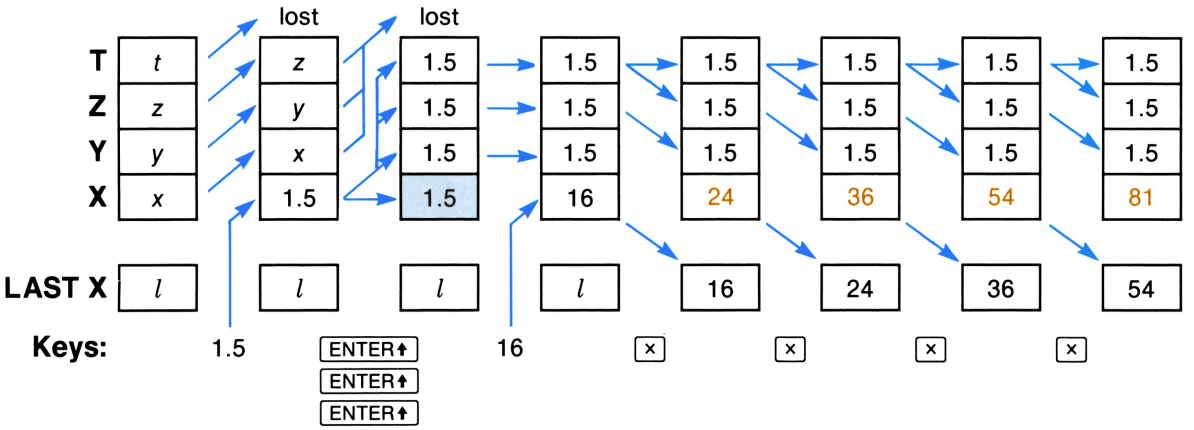
Noncommutative Functions. Functions like subtraction and division are called *noncommutative* because the order of the operands is essential: $5 - 3 \neq 3 - 5$, and $5 \div 3 \neq 3 \div 5$. For expressions involving noncommutative functions, enter or calculate the number that must be in the Y-register 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.



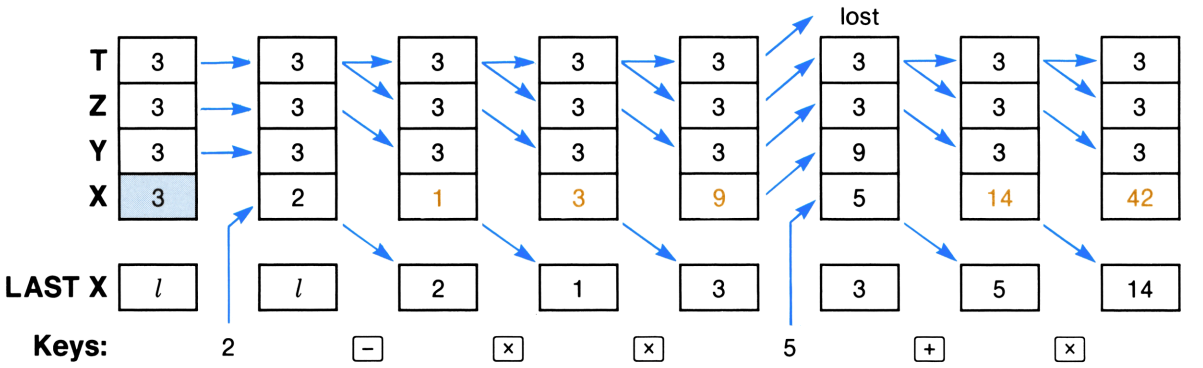
Polynomial Expressions. Filling the stack aids the evaluation of a polynomial, which requires several copies of the variable. For efficiency, use Horner's Method to rewrite the polynomial in a nested fashion that eliminates exponents greater than 1. Suppose that you want to evaluate

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

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:

$$a, [\times], b, [+], [\times], c, [+], [\times], d, [+], [\times], e, [+], [\times], f, [+].$$

- 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

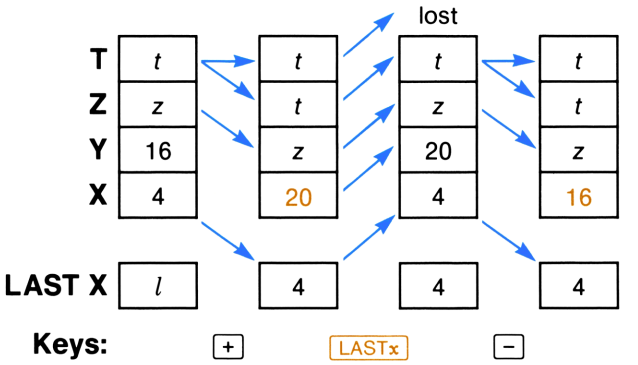
The LAST X register holds the x-operand from the last numeric function (except [CHS]). To recall this number to the X-register, press [LASTx]. This enables you to recover from errors and to retrieve an operand for further calculations.

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.

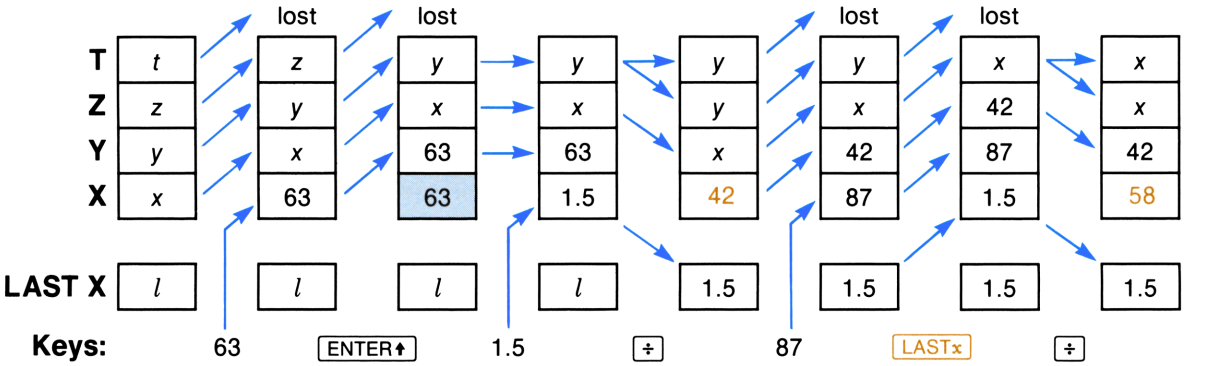


- If you wanted to multiply instead of add, execute [LASTx] again to return the stack to its original state, and then multiply.
- If 16 was the wrong number to add, press [←] to clear 16, key in the correct number, execute [LASTx] to recover 4, and then add.
- If 4 was the wrong number to add, key in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

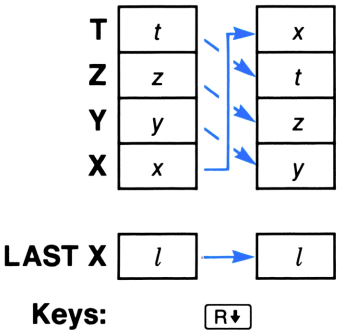
Exchanging the X- and Y-registers. Executing [x≷y] (*X exchange Y*) exchanges the contents of the X- and Y-registers. This function has several uses:

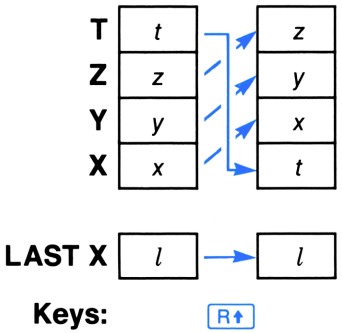
- To examine the contents of the Y-register. Press [x≷y], examine the display, and then press [x≷y] again to restore the numbers to their original order. This is useful when a function returns results to both the X- and Y-registers, as do the statistics functions and polar/rectangular coordinate conversions.
- 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

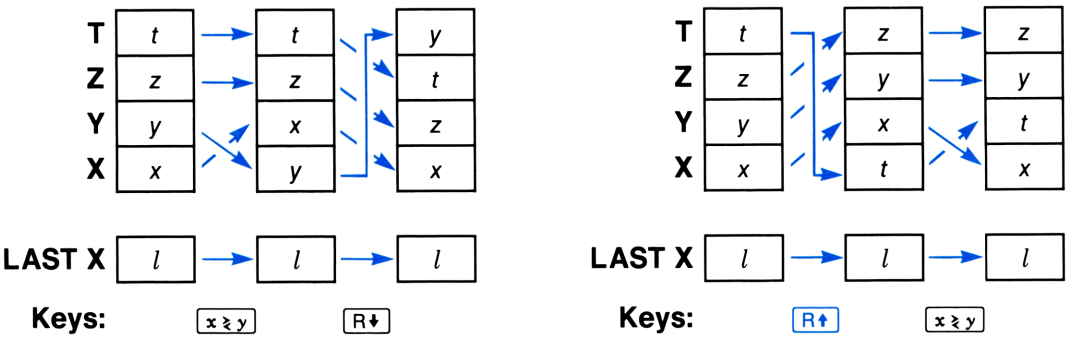
The [R↓] (*roll down*) and [R↑] (*roll up*) functions shift all stack contents without duplicating or losing any data.





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.

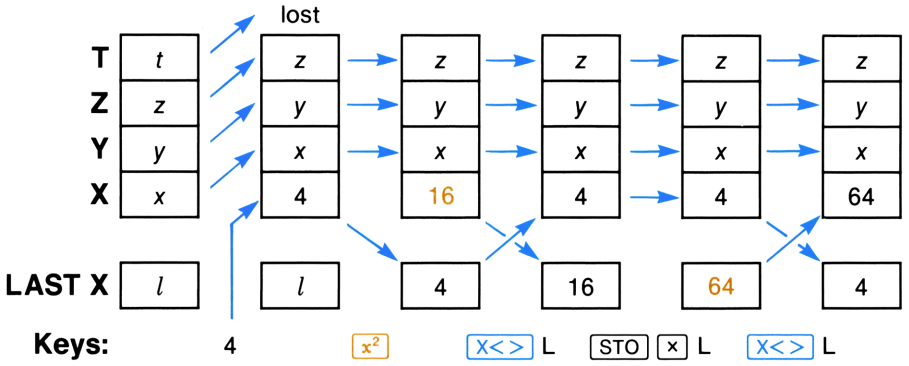
Recall. To copy the number in a stack register or the LAST X register into the X-register, press [RCL] [·] followed by [Y], [Z], [T], or [L]. The number in the T-register is lost as the stack lifts (unless stack lift is disabled).

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 11

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.

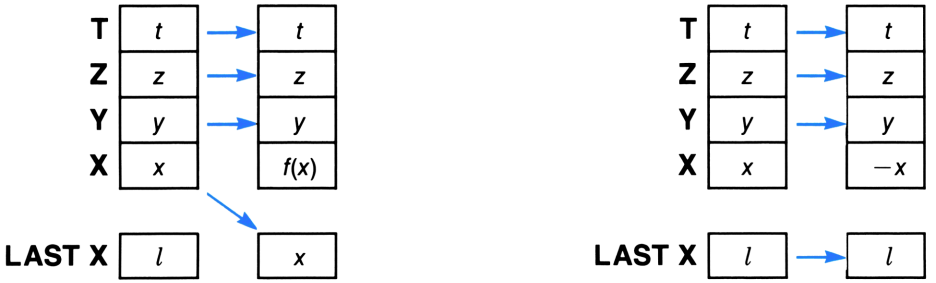
There are three error conditions that can result from numeric functions.

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 [x²] returns the square of x. Executing [√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:

$$1 \text{ if } x \geqslant 0,$$

$$-1 \text{ if } x < 0, \text{ or}$$

$$0 \text{ if the X-register contains Alpha data.}$$

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,

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

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,

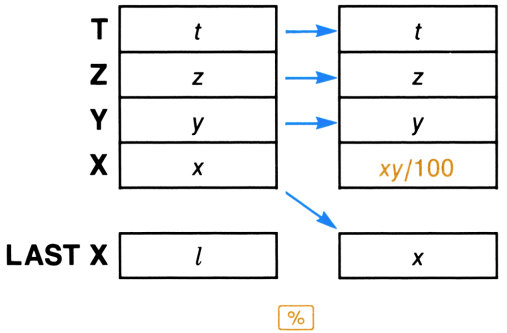
- Subtraction returns $y - x$ (not $x - y$).
- Division returns y/x (not x/y).

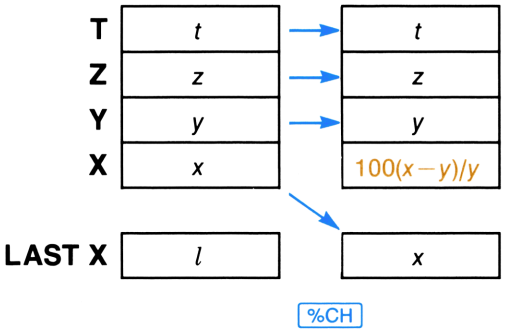
Time Arithmetic

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:

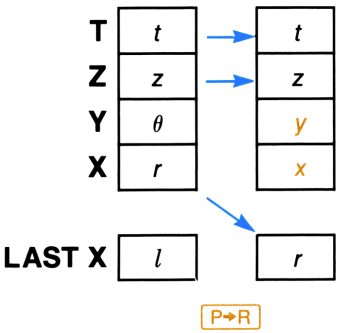
1. Place the total (base) number in the Y-register and the number to be converted to a percentage in the X-register.
2. Execute [1/x].
3. Execute [%].
4. Execute [1/x].

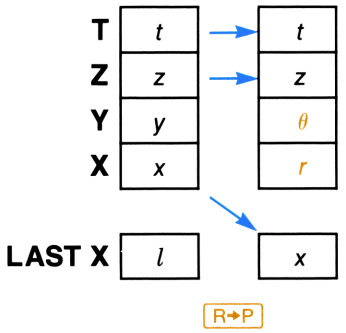
Polar/Rectangular Conversions

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 54.) 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.





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 on pages 55 through 58.

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 check the current assignment, execute [ΣREG?]. The address of the first register in the block is returned to the X-register.*
- To place zeros in all six statistics registers, execute [CLΣ].

* [ΣREG?] doesn't check whether the assigned registers actually exist (that is, whether sufficient memory is allocated to data storage), but only what the assigned address is.

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≷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≷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.

Part III:
Memory in Detail

Section 12

Main Memory

Contents

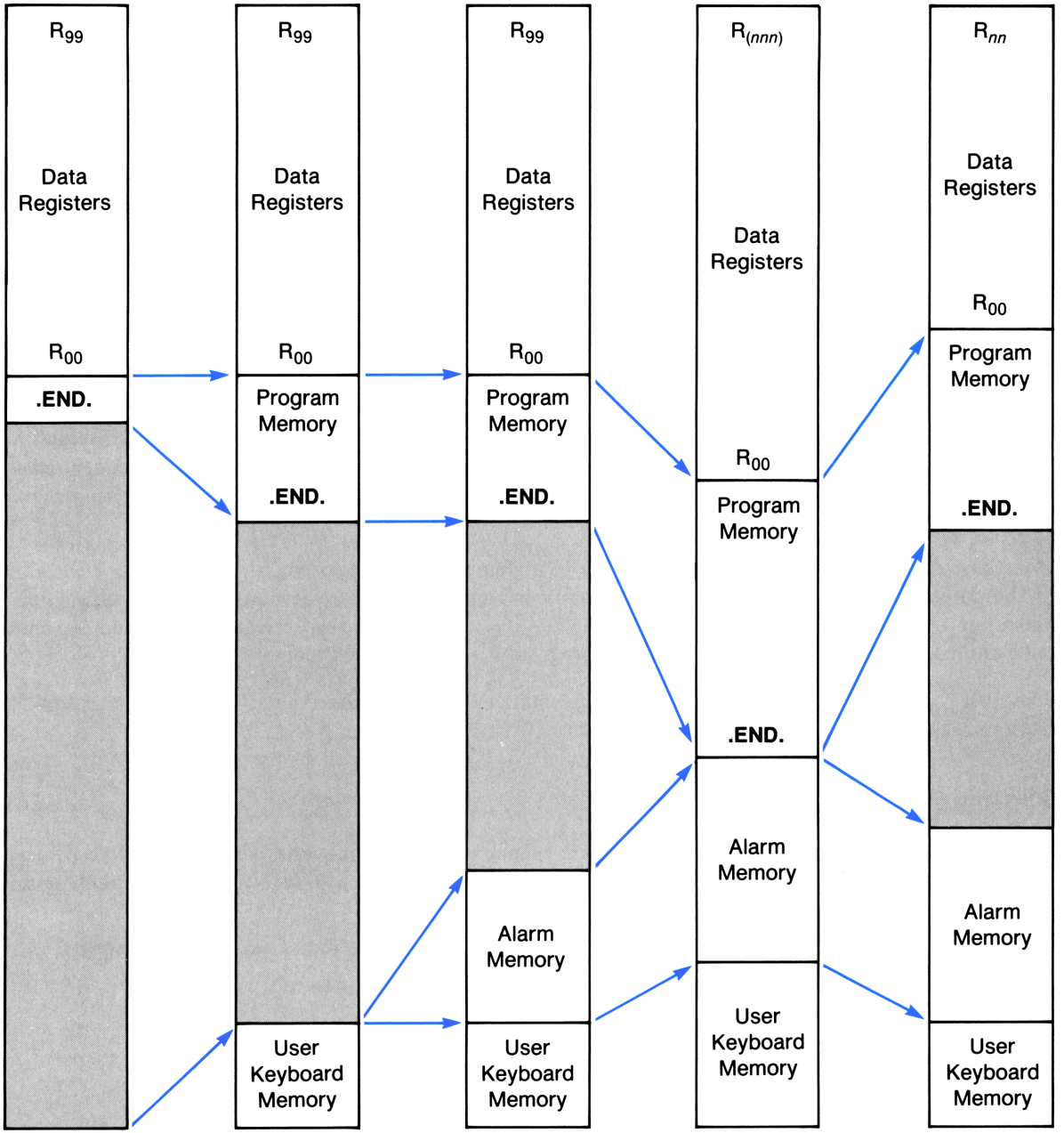
Organization

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, alarms, key redefinitions, and uncommitted registers. The uncommitted registers are automatically committed to programs, alarms, 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 five configurations of main memory—each column represents all of main memory at one time. The leftmost column shows the default configuration, 100 registers allocated to data storage and 219 registers for all other purposes. The columns to the right show main memory at four 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 100 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, alarms, 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 alarm memory after you have set alarms. The fourth column shows the result after you allocate more registers to data storage, write more programs, set more alarms, and redefine more keys. When all registers are committed, any operation that would consume main memory registers causes an error. If you are setting an alarm, the computer displays **NO ROOM**. Other operations cause 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, alarm, or User-keyboard memory by executing [CATALOG] 1, 5, or 6 respectively; also remember that data and programs can be stored in data or program files in extended memory.

The fifth column shows the reappearance of uncommitted registers after you allocate fewer registers to data storage and delete other memory contents.

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.** ***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 programs by executing CLP or PCLPS.
- 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.

Alarm Memory

When you set the first alarm, two uncommitted registers are consumed to define the limits of alarm memory. (These registers are in addition to the registers consumed by the alarm itself.) They will be released only when all alarms are cleared. The number of registers consumed by each alarm depends on the type of alarm.

- Each alarm uses one register for the time and date of its next activation.
- An alarm with a reset interval requires one additional register.
- An alarm that uses the Alpha register requires one register for every 7 characters. († is an ordinary character for counting purposes.) For example, a message of 18 characters requires two registers for the first 14 characters and one register for the last 4 characters, for a total of three registers.

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 by one of the methods on page 168. 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. There are two functions that allocate main memory registers to data storage, one for manual execution and one for program execution. Decreasing the number of registers loses the data in the largest-numbered registers.

- You can manually change the allocation to data storage by executing [SIZE] and then specifying the number of registers to be allocated.
- A program can change the allocation to data storage by placing the number of registers to be allocated in the X-register and then executing [PSIZE].

Checking the Allocation. To check the current allocation of registers to data storage, execute [SIZE?]. The total number of data storage registers is returned to the X-register.

Example. The following program routine will change the allocation only if the current allocation is too small.

01 SIZE?	Returns the current allocation to the X-register.
02 50	The number of data registers required by the program. The current allocation (from line 01) is now in the Y-register.
03 X>Y?	Is the required number of data registers greater than the current allocation?
04 PSIZE	If so, change the allocation to the required number.

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. Unless you want to use indirect addressing in a loop or to perform special main memory operations like register arithmetic, it's often easier to use a data file in extended memory.

Data Register Operations

Store and Recall

There are three sources/destinations for the data in data registers: the stack registers, the Alpha register, and data files in extended memory. 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. All functions that access data files in extended memory are discussed in section 13, "Extended Memory."

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 [f] 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.
- If you do want to perform numeric operations on Alpha digits in the Alpha register, you must use [ANUM] to interpret the digits as a number and place the number in the X-register (page 311).
- 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.
- To rotate the six characters you already copied to the right-hand end of the Alpha register, place 6 in the X-register and execute [AROT] (*Alpha rotate*).

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.

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.

Block Operations

You can move or swap blocks of data registers by using [REGMOVE] or [REGSWAP]. Both use a control number *sss.dddnnn* in the X-register to define the source and destination blocks, where:

- R_{sss} is the first (smallest-addressed) register in the *source* block;
- R_{ddd} is the first (smallest-addressed) register in the *destination* block;
- *nnn* is the *number* of registers in both blocks. If you don't specify *nnn*, the computer assumes *nnn* = 001.

The blocks can't overlap—make sure that $|sss - ddd| \geqslant nnn$.

Moving Register Contents. To copy the source block into the destination block, place the control number in the X-register and execute [REGMOVE]. As with [STO], the contents of the source block are unchanged and the previous contents of the destination block are lost.

Swapping Register Contents. To exchange the contents of two blocks of registers, place the control number in the X-register and execute [REGSWAP]. Which block is called the source and which is called the destination is immaterial.

Clearing Registers

Clearing a Single Register. To clear a single register, store zero in that register.

Clearing a Block of Registers. It's wise to begin a program by clearing the data registers used by that program. To clear a block of registers, execute CLRGX (*clear registers by X*) with a control number in the X-register. The control number has the form *bbb.eeeii* where:

- R_{bbb} (*begin*) is the first (smallest-addressed) register to be cleared;
- R_{eee} (*end*) is the last (largest-addressed) register to be cleared;
- *ii* is the *increment* if you want only every *ii*th register cleared. If you don't specify *ii*, the computer assumes $ii = 01$.

As an example of using the optional increment, suppose you have a 4×4 matrix stored sequentially in memory, row by row. You can clear any row of the matrix by specifying just the first and last addresses in the row—you don't need to specify an increment. But to clear a column (whose entries aren't sequential in memory), first specify the addresses for the first and last entries in that column, and then set $ii = 04$ to clear only the registers for that column.

R_{bbb} is cleared even if $bbb > eee$ or $bbb + ii > eee$. The sign of the control number and any excess fractional digits are ignored.

Clearing All Registers. To clear all data registers, execute CLRG.

Section 13

Extended Memory

Contents

Introduction

Extended memory is an extension of main memory, but it also has special advantages of its own. All information in extended memory is organized into files; there are three types of files for three types of information.

- A program in main memory can be stored in a program file. You must return the program to main memory to execute it or edit it.
- Register contents can be stored in a data file, which is a collection of registers. A single data-file operation can access all data registers in main memory, a block of data registers, or the X-register. Directly accessing the X-register makes extended memory an alternative to main memory as well as an extension of it.
- Alpha data can be stored in a text file (also called an ASCII file), which is a collection of Alpha strings. The size and structure of text files, along with direct keyboard access to them through a text editor, give text files an Alpha capability far surpassing main memory.

Because each file requires two registers for the computer's use, the number of extended memory registers available for your use depends on the number of files. A single file can contain up to 124 registers for your use; two files can contain up to 122 registers for your use; in general, n files offer $126 - 2n$ registers for your use.

You can increase the number of extended memory registers by adding one or two HP 82181A Extended Memory Modules. (These modules are described in appendix E.) Each module adds 238 registers. The maximum number of extended memory registers is therefore $124 + 238 + 238 = 600$, almost twice as many registers as in main memory.

Files in Extended Memory

Header Information

Each file starts with a *header*, two registers that describe the file. Although you never access headers directly, you should be aware of the memory space and the information involved. A header contains:

The File Name. You give each file a name when you create the file. The name can include any characters except commas and a maximum of seven characters.* You enter the file name into the Alpha register when you create the file and when you specify the file.

The File Type. A file may be a program file, a data file, or a text file. A program file is a copy of a program in main memory; a data file is a collection of data registers; and a text file is a collection of character strings.

---

* A file name can't consist of seven bytes of decimal value 255.

The File Size. This is the number of extended-memory registers the file requires. For program files the header also contains the program size in bytes.

The Pointer. Data and text files use a number called a pointer to indicate which element in the file to access. Program files do not use pointers.

Specifying a File

You can access only one file at a time. The computer keeps track of which file you accessed last; this file is called the *current file*. Only some extended memory functions can specify which file to access.

- Some functions *require* a file name in the Alpha register. Program-file functions and destructive functions are examples.
- Some functions *can specify* a file name in the Alpha register. If the Alpha register is empty, these functions act on the current file.
- Some functions *never specify* a file name in the Alpha register. These functions act only on the current file.

Extended Memory Directory

Both [EMDIR] (*extended memory directory*) and [EMDIRX] (*extended memory directory by X*) are programmable functions that access the files in extended memory. The function [EMDIR], which lists all files, is designed for interaction with the user; the function [EMDIRX], which returns information about one specified file, is designed for automatic operations in a program.

Using [EMDIR]. For convenience you can manually execute [EMDIR] as catalog 4. If you want to assign it to a key or enter it as a program line, you must use the function's Alpha name [EMDIR]. When you execute [EMDIR]:

- The files are listed in the order that you created them.
- Each file's name, type (P = program, D = data, A = ASCII or text), and number of registers are shown. For example, the display for a data file ABC that occupies 20 registers is:

```
ABC      D020
```

- You can speed up the listing by holding down any key except [R/S] or [ON].
- You can stop the listing by pressing [R/S], and restart the listing by pressing [R/S] again.
- When the listing is stopped you can go to the next file by pressing [SST] or return to the previous file by pressing [BST]. Pressing [SST] while the last file is displayed or [BST] while the first file is displayed has no effect.

You can either allow the computer to complete the listing and exit [EMDIR] automatically, or else you can terminate [EMDIR] yourself. If you allow the listing to run past the last file, the computer automatically exits [EMDIR] and returns the number of registers available for the next file you create. The two registers required for a new header are included in this calculation, so you can use the total number of registers indicated for your own data. This number is returned to the X-register, lifting the stack unless stack lift is disabled.

When the computer automatically exits [EMDIR] it doesn't change which file is the current file. Alternatively, you can make any file the current file by stopping the listing with that file displayed and then pressing [←]. This terminates [EMDIR]; the number of available registers is *not* returned to the X-register.

If there are no files in extended memory the computer displays **DIR EMPTY** and returns the number of available registers to the X-register, lifting the stack unless stack lift is disabled.

Using [EMDIRX]. To determine the name and type of the *n*th file in extended memory, place *n* in the X-register and execute [EMDIRX]. For example, $n = 1$ for the file you created first (which is also the file displayed first by [EMDIR]).

If the *n*th file exists:

- The name of the *n*th file is returned to the Alpha register.
- The file type is returned to the X-register as a two-letter code:*

  PR = program file.
  DA = data file.
  AS = ASCII (text) file.
- *n* is placed in the LAST X register.
- The *n*th file becomes the current file.

If the *n*th file doesn't exist:

- The Alpha register is cleared.
- Zero is returned to the X-register.
- *n* is placed in the LAST X register.
- The computer doesn't change which file is the current file.

Note that [EMDIRX] does *not* lift the stack. The input *n* is the absolute value of the integer part of the number in the X-register.

* A positive integer is returned if the computer doesn't recognize the file type.

Using EMROOM**.** To check how many registers are available for the next file you create, execute EMROOM (*extended memory room*). The result takes into account two registers for one new header; if you plan to create more than one new file, subtract two registers for each additional file. The result is returned to the X-register, lifting the stack unless stack lift is disabled. (This is the same information that EMDIR provides if it exits automatically.)

Checking File Size

To check the number of registers in a file (excluding the header), place the file name in the Alpha register and execute FLSIZE (*file size*). The result is returned to the X-register, lifting the stack unless stack lift is disabled. You can check the size of the current file by clearing the Alpha register and executing FLSIZE.

Purging a File

To remove a file from extended memory—both its header and its contents—place the file name in the Alpha register and execute PURFL (*purge file*). All later files in extended memory move up to fill the space previously held by the purged file.

There is no current file after you execute PURFL, so the next extended memory function executed *must* specify a file name to define a new current file.

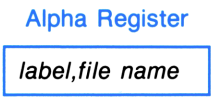
Program File Operations

Program files are the simplest type of extended memory file. A program file comprises a two-register header plus an exact copy of a program from main memory.

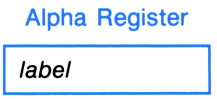
Saving a Program

You can create a file that is a copy of a program in main memory by first specifying the program to be copied and the new file's name in the Alpha register and then executing SAVEP (*save program*). There are three ways to specify the program and name the new file:

- Indicate the program to be copied by any global label in that program, followed by a comma, followed by the name you want to give the new file.



- If you want to use the global label for the name of the new file, you can place only the global label in the Alpha register.



- If the program to be copied is the current program in main memory, you can place just a comma and the name of the new file in the Alpha register.

If a global label is in the Alpha register, [SAVEP] copies the program closest to the bottom of program memory (that is, listed last in catalog 1) containing that global label.

The file name can include any characters except commas and a maximum of seven characters. The eighth and all subsequent characters, or a comma and all subsequent characters, will be ignored.

> **Note:** If the new file name (whichever of the three ways it is defined) is identical to the name of an existing program file, the existing program file is overwritten.

Getting a Program

There are two functions, [GETP] (*get program*) and [GETSUB] (*get subroutine*), that copy a program from extended memory to main memory. The functions both place the copied program at the bottom of program memory, but they differ in other respects.

- [GETP] *deletes* the program that was at the bottom of program memory and (in some cases) makes the first line of the recalled program the current program line.
- [GETSUB] *retains* the program that was at the bottom of program memory and never alters the current program line.

Keyboard Execution.

- If you want to replace the program at the bottom of program memory, place the name of the program file in the Alpha register and execute [GETP]. The first line of the recalled program becomes the current program line; press [R/S] to run the program.
- If you want to run the recalled program by pressing [R/S] but you want to save the program at the bottom of program memory, press [GTO] [.] [.] before you execute [GETP]. (The null program created by [GTO] [.] [.] will be replaced.)
- If you want to save the last program, and especially if you're recalling several programs at once, place the name of the program file in the Alpha register and execute [GETSUB]. This doesn't change which program line is the current program line.

Program Execution. GETP operates in two different ways, depending on whether the program containing GETP is at the bottom of program memory; GETSUB operates identically in either case. This means that there are three options for a program to recall another program from extended memory:

- If a program executes GETSUB, the recalled program is placed at the bottom of program memory and execution continues with the instruction following GETSUB.
- If a program *not* at the bottom of program memory executes GETP, the recalled program replaces the program at the bottom of program memory and execution continues with the instruction following GETP.
- If a program at the bottom of program memory executes GETP, that program is replaced by the recalled program and execution continues with the first line of the recalled program.

Examples. The three options above give you great flexibility to recall a program from another program, and in many situations there is more than one option that will work. These examples illustrate situations in which you would select a particular option. Suppose that MAIN is a program in main memory (but not at the bottom) and that AA, BB, and CC are programs in extended memory that MAIN calls as subroutines.

The first four examples illustrate the first two options, using GETSUB and GETP in the program MAIN. To ensure that MAIN isn't at the bottom of program memory you could press GTO . . before executing MAIN (to create a null program), or MAIN could first use GETSUB to recall a dummy program containing only a global label and an END instruction. (Recalling a dummy program could also enable MAIN to conclude with PCLPS, clearing the dummy program and all subroutines recalled by MAIN.)

- If MAIN calls AA, BB, and CC as subroutines and there is enough room in main memory for all three programs: use GETSUB three times at the start to recall AA, BB, and CC, and use XEQ as needed to call each subroutine.
- If there isn't enough room for AA, BB, and CC at once (and MAIN always calls them in that order): use GETSUB and XEQ to call AA, and use GETP and XEQ to call BB and CC.
- If there isn't enough room for AA, BB, and CC at once (and you don't know in advance in which order MAIN will call them): use GETSUB, XEQ, and PCLPS for each subroutine call; or place a null program or dummy program at the bottom of program memory and then use GETP and XEQ for each subroutine call.
- If AA contains a second global label DD and you want the subroutine call to begin at DD: use GETSUB to recall AA and then use XEQ DD to start the subroutine at DD.

The next two examples illustrate the first and third options, using GETSUB and GETP in the subroutine AA. Assume that MAIN has recalled AA to the bottom of program memory.

- If AA branches to BB, use GETP in AA to recall BB. This automatically clears AA and transfers execution to BB.
- If AA calls BB as a subroutine, use GETSUB and XEQ in AA to call BB. This leaves AA intact in program memory so that execution can return to AA after BB finishes.

Automatic Key Redefinition

Recall from section 10 that a global label assigned to the User keyboard contains that assignment information. (This is true only for programs listed in catalog 1.) When SAVEP copies a program into extended memory, any current assignments are saved along with the assigned global labels. If the User keyboard is active (that is, if flag 27 is set) when GETSUB or GETP recall a program from extended memory, any assignments stored with that program will be reactivated.

Checking Program Size

You can check how many bytes are in a program stored in a program file by using RCLPT or RCLPTA. (These functions are primarily used with data and text files; this application with program files is distinct from their primary purpose.) Both functions return the number of bytes to the X-register, lifting the stack unless stack lift is disabled.

- If the program file is not the current file, place its name in the Alpha register and execute RCLPTA.
- If the program file is the current file, execute RCLPT or else clear the Alpha register and execute RCLPTA.

Creating Data and Text Files

You must explicitly allocate space in extended memory for each data or text file. Before creating a file you can check how many registers are available by executing EMDIR (also available as CATALOG 4) or EMROOM. The number returned to the X-register is the maximum number of registers you can allocate to a new file.

Creating Data Files

Use CRFLD (*create file/data*) to create a data file as follows.

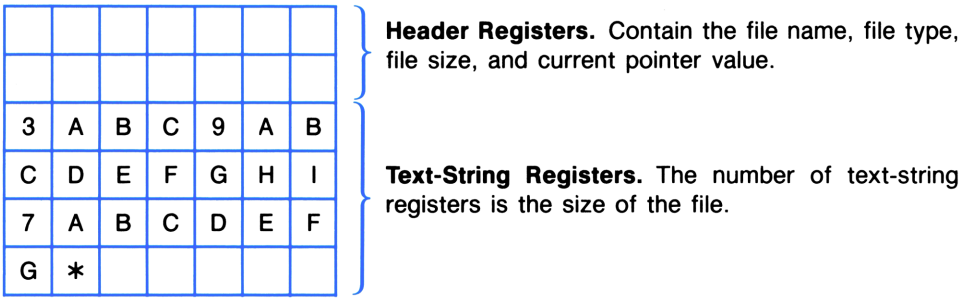
1. Place the name of the new file in the Alpha register. The name can include any characters except commas and a maximum of seven characters.
2. Place the number of registers you want to allocate to the file in the X-register. (Do not include registers for the header.)
3. Execute CRFLD.

This new file is now the current file, and its pointer is set to the first register.

Creating Text Files

Text files are created similarly to data files, but first you must translate the memory requirements of your file into registers. A rough estimate is usually sufficient because you can adjust the size later; the exact formula given here can be useful when a program creates the file.

Shown below is a text file of three records: ABC, ABCDEFGHI, and ABCDEFG. At the start of each record is an extra byte (like XTOA creates) that indicates the length of that record. A record can contain up to 254 characters.



The "*" at the end of the file is a byte that indicates the end of the current contents of the file. (The decimal value for this byte is 255; this is why each record must have a length less than 255.) This byte and the bytes indicating record lengths are invisible to you; the file looks like the diagram on page 215 when you use it. The only time you must consider these extra bytes is when you create a text file.

If you know exactly how many records and characters will be in the file, you can calculate exactly how many registers are required.

1. Add the number of records to the number of characters in all records.
2. Add 1 to the result.
3. Divide the result by 7. If there is any remainder, round up to a whole number.

Now use CRFLAS (*create file/ASCII*) to create the file.

4. Place the name of the new file in the Alpha register. The name can include any characters except commas and a maximum of seven characters.
5. Place the number of registers to be allocated in the X-register.
6. Execute CRFLAS.

The new file is now the current file, and its pointer is set to the first character in the first record.

Resizing Files

You can change the number of registers allocated to a data or text file by executing RESZFL (*resize file*) as follows:

1. Make sure that the file to be resized is the current file.
2. Place in the X-register the number of registers to be allocated.
3. Execute RESZFL.

The integer part of the number in the X-register becomes the new size of the current file. The pointer is unchanged.

Increasing File Size. Before enlarging a file you can check how many registers are available by executing EMDIR or EMROOM. You can usually add two registers more than the result, because EMDIR and EMROOM reserve two registers for a new header that isn't required in this case. However, if the number returned is zero, there might be two, one, or no registers left.

Reducing File Size. If you reduce the size of a file by *n* registers, the last *n* registers in the file are lost. With a *positive* number in the X-register, RESZFL executes only if no registers currently in use will be lost. A register is considered in use if it:

- Contains a non-zero number (for data files).
- Contains characters or the end-of-file byte (for text files).

If the number in the X-register is positive and registers in use would be lost, RESZFL causes a **FL SIZE ERR**. For data files you can bypass this protection by a *negative* number in the X-register. In this case, the last *n* registers are lost regardless of their contents.

Clearing Files

You can clear the contents of a data or text file without purging the header as well by executing CLFL (*clear file*) as follows:

1. Place the name of the file in the Alpha register.
2. Execute CLFL.

The cleared file becomes the current file. If it's a data file, all registers contain zero and the pointer is set to the first register. If it's an text file, no records exist and the pointer is set to the first character in the first record to be added.

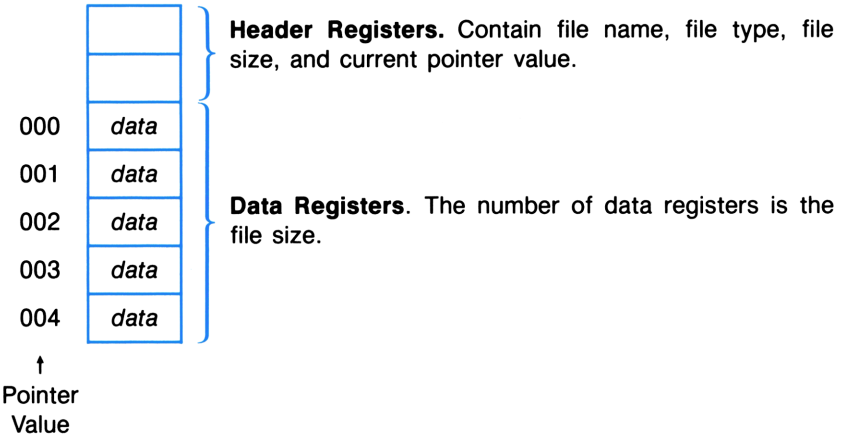
Pointers in Data and Text Files

Data and text files are collections of registers and Alpha strings, respectively, with pointers that enable you to access individual elements in those collections. The structures of data and text files are described first, followed by pointer operations.

Structure of Data Files

A data file comprises two header registers and one or more data registers. The header contains the current pointer value, which indicates the register that will be accessed next.

The diagram below shows a data file of five registers. Note that the pointer values start with 000 for the first register and are greater by one for each subsequent register.



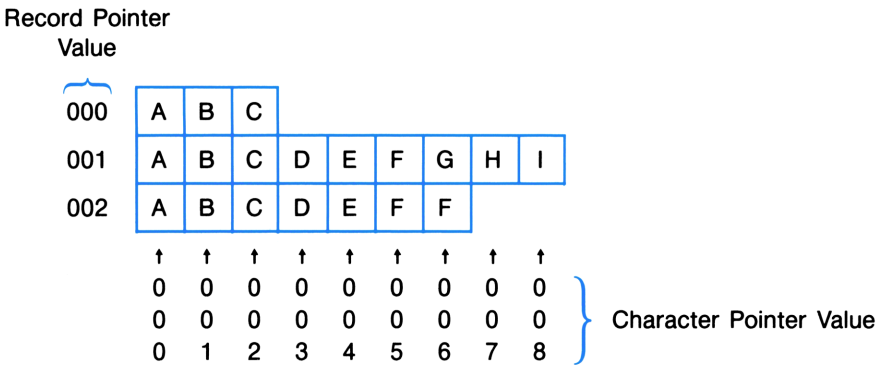
Structure of Text Files

A text file comprises two header registers and one or more registers that contain your records. Each record is a string of up to 254 characters. This means that records can be different lengths, and that the pointer for a text file (contained in the header) has two components:

Record Pointer. The integer part of a text-file pointer is the record pointer. Its value indicates the current record.

Character Pointer. The fractional part of a text-file pointer is the character pointer. Its value indicates the current character within the current record. Functions that act on an entire record ignore this part part of the pointer.

A text-file pointer is represented as *rrr.ccc*, where *rrr* is the record pointer and *ccc* is the character pointer. In the diagram below, which shows a text file containing three records, the pointer values for the three A's are 000.000, 001.000, and 002.000. The pointer values for H and I are 001.007 and 001.008. (This diagram illustrates how to interpret text-file pointers. The diagram on page 212 shows the same file as it would actually be stored in registers.)



Pointer Operations

Pointers allow you to access individual elements within data and text files. The current value of a pointer indicates the current register or the current record and character; the pointer must be set *before* you execute the function that accesses that register, record, or character.

Often you will explicitly set a pointer when you first access a file, preparing for the subsequent functions that will access the indicated data. Most functions that use pointers also automatically advance the pointer. This makes it possible to access sequential elements simply by repeating the function.

There are times when the value of a pointer doesn't correspond to any stored information. For example, if a function accesses the last register in a data file and then advances the pointer, the new pointer value doesn't correspond to a register in the file. If you repeated that function, an **END OF FL** error would result.

Setting a Pointer. To set a pointer, execute SEEKPTA (*seek pointer by Alpha*) or SEEKPT (*seek pointer*).

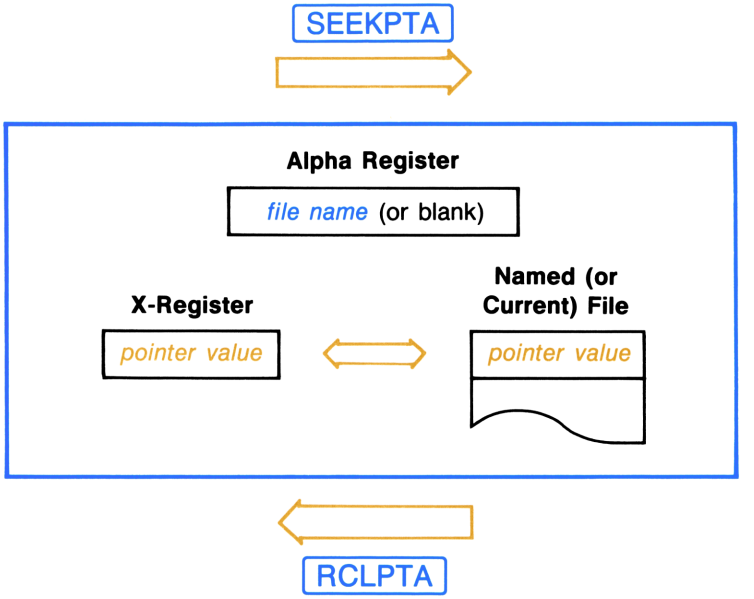
If the desired file is not the current file, use SEEKPTA.

1. Place the name of a data or text file in the Alpha register.
2. Place the desired pointer value in the X-register.
3. Execute SEEKPTA.

The file named in the Alpha register becomes the current file, with its internal pointer set to the number in the X-register. For a data file, the integer part of this number sets the register pointer. For a text file, the integer part sets the record pointer and the first three digits of the fractional part set the character pointer.

If the desired file is already the current file, you can either:

- Place the pointer value in the X-register and then execute SEEKPT; or
- Clear the Alpha register, place the pointer value in the X-register, and then execute SEEKPTA.



Checking a Pointer Value. You can check the current value of a pointer by using RCLPTA (*recall pointer by Alpha*) or RCLPT (*recall pointer*). If the desired file is not the current file, use RCLPTA as follows:

1. Place the name of the file in the Alpha register.
2. Execute RCLPTA.

The file named in the Alpha register becomes the current file. Its current pointer value is returned to the X-register, lifting the stack unless stack lift is disabled. For a data file, the number represents the register pointer. For an text file, the number has the form *rrr.ccc*, where *rrr* is the record pointer and *ccc* is the character pointer.

You can check the current pointer value for the current file by executing RCLPT or by clearing the Alpha register and executing RCLPTA. The pointer value is returned to the X-register as described above. RCLPTA and RCLPT also have a secondary use with program files, described on page 211.

Data File Operations

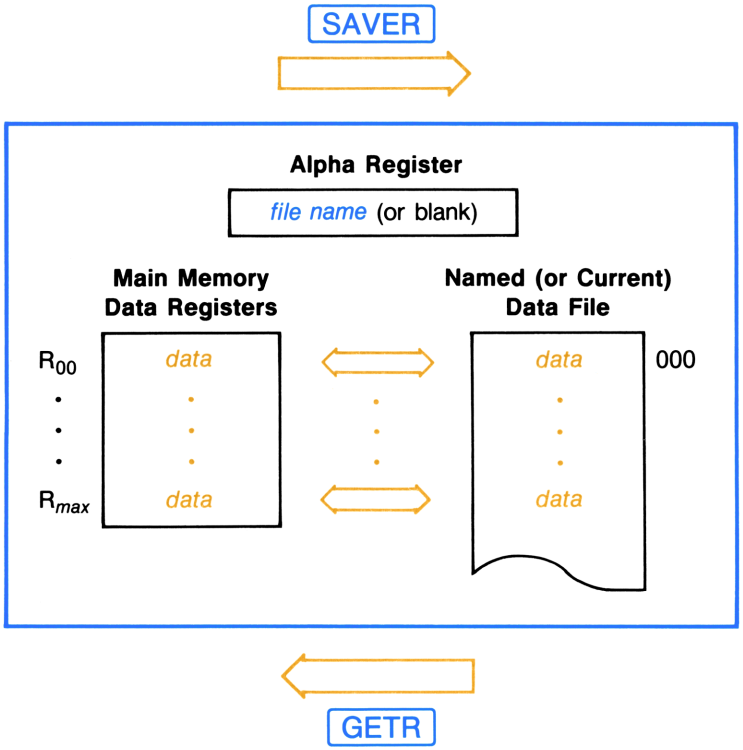
Data files are useful as an extension of main memory and as an alternative to main memory. For operations such as register arithmetic and indirect addressing, your data must be in main memory; but there are also extended memory operations that can't be duplicated in main memory. The functions that access data registers in main memory are described first, followed by the functions that access the X-register.

Accessing All Data Registers

You can copy all data registers in main memory into a data file by using [SAVER] (*save registers*), and you can copy all or some of a data file into main memory by using [GETR] (*get registers*). In either direction, data moves between registers with corresponding addresses and pointer values: between R_{00} in main memory and register 000 in the data file, between R_{01} and register 001, and so on. The current value of the pointer doesn't affect these functions, but both functions advance the pointer just past the last file register accessed.

Copying Data From All Main Memory Data Registers. To copy each data register in main memory into the corresponding register in a data file, place the name of the data file in the Alpha register and then execute [SAVER]. If the desired file is already the current file, you can simply clear the Alpha register and execute [SAVER].

If there are fewer destination registers (in the data file) than source registers (in main memory), no registers are copied and an **END OF FL** error occurs. This is because [SAVER] expects to save the data in *all* main memory registers.



Copying Data Into All Main Memory Data Registers. To copy the registers in a data file into the corresponding data registers in main memory, place the name of the data file in the Alpha register and then execute [GETR]. If the desired file is the current file, you can simply clear the Alpha register and then execute [GETR].

Starting with file register 000 and main memory register R_{00}, [GETR] moves data between corresponding registers until there are no further registers of one type or the other. (This is unlike [SAVER], which causes an error if not all main memory data registers are accessed.)

Accessing a Block of Data Registers

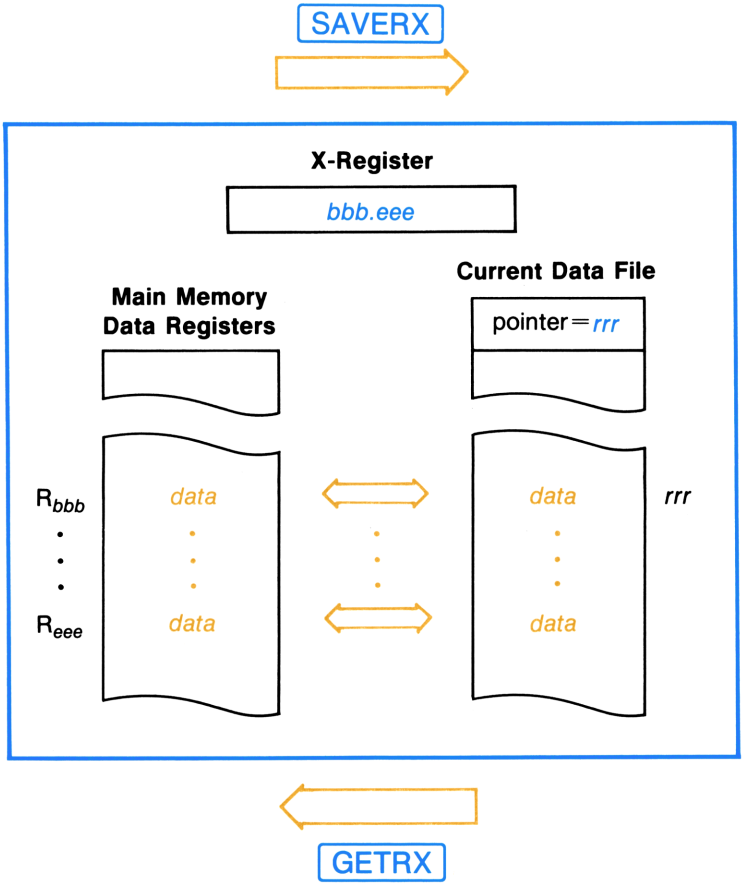
You can move data between a block of data registers in main memory and a block of the same size in a data file by using [SAVERX] (*save registers by X*) and [GETRX] (*get registers by X*). The data file must be the current file. The two blocks of registers are defined as follows:

- The block of registers in main memory is defined by a control number in the X-register. The control number has the form *bbb.eee*, where R_{bbb} is the register that *begins* the block and R_{eee} is the register that *ends* the block.
- The block of registers in the data file is defined by the current pointer value and by the control number in the X-register. The block begins with the current register and includes the *eee* − *bbb* registers that follow.

The data move between R_{bbb} in main memory and the current register in the file, between R_{bbb+1} and the next register in the data file, and so on. Both [SAVERX] and [GETRX] advance the data file's pointer just past the last register accessed. If there are fewer file registers (from the current register to the end of the file) than specified for the block in main memory, no registers are copied and an **END OF FL** error occurs.

Copying Data From a Block of Main Memory Data Registers. Use [SAVERX] to copy the contents of a block of registers in main memory into a block of registers in a data file:

1. Be sure that the destination file is the current file and that its pointer is set to the first register to receive data.
2. Place the control number *bbb.eee* in the X-register.
3. Execute [SAVERX].



Copying Data Into a Block of Main Memory Data Registers. Use GETRX to copy the contents of a block of registers in a data file into a block of data registers in main memory:

1. Be sure that the source file is the current file and that its pointer is set to the first register to send data.
2. Place the control number *bbb.eee* in the X-register.
3. Execute GETRX.

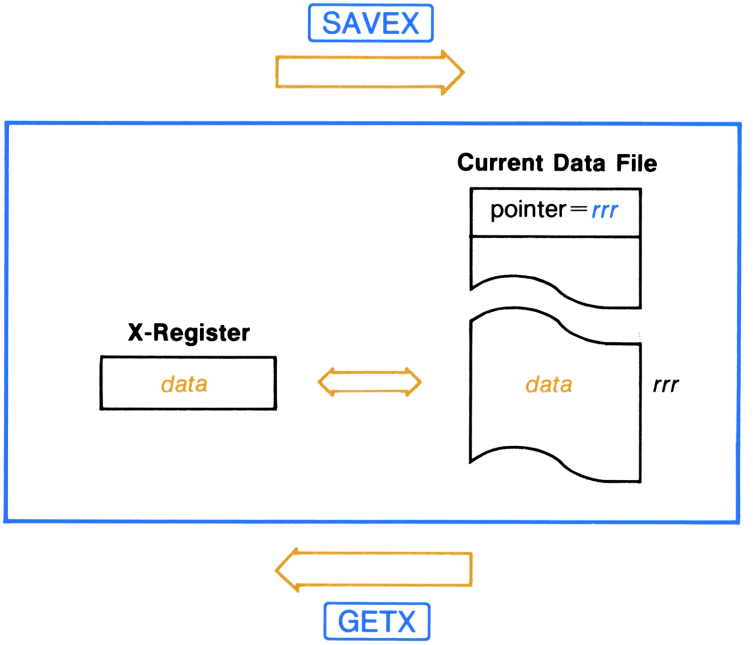
Accessing the X-Register

You can move data between the X-register and a data file by using [SAVEX] (*save X*) and [GETX] (*get X*). The data file must be the current file and the register to be accessed must be the current register. Both functions advance the data file's pointer just past the register accessed.

Storing the Contents of the X-register. Use [SAVEX] to copy the contents of the X-register into a data-file register:

1. Be sure that the destination file is the current file and that the destination register is the current register.
2. Execute [SAVEX].

The X-register is unchanged and the previous contents of the destination register are lost. (This is similar to [STO].)



Recalling Data to the X-register. Use [GETX] to copy the contents of a data-file register to the X-register:

1. Be sure that the source file is the current file and that the source register is the current register.
2. Execute [GETX].

The source register is unchanged; the stack lifts and the contents of the T-register are lost, unless stack lift is disabled. (This is similar to [RCL].)

Example. Suppose that a program needs to store the stack contents before using the stack to set an alarm, and then later needs to return those contents to the stack. This is done in three stages, listed below as separate routines.

The first routine creates a data file named STACK; it is executed at the beginning of the program, before the stack contains the values to be stored.

01 ^T STACK	Places the file name in the Alpha register.
02 4	Places the number of registers in the X-register.
03 CRFLD	Creates a data file named STACK containing four registers.

The second routine copies the numbers in the stack into the file; it is executed just before placing the alarm parameters in the stack.

01 ^T STACK	Places the file name in the Alpha register.
02 STO L	Saves the contents of the X-register in the LAST X register.
03 CLX	Places zero in the X-register (without lifting the stack).
04 SEEKPTA	Makes STACK the current file and makes register 000 the current register.
05 X<> L	Returns the contents saved in the LAST X register to the X-register (without lifting the stack).
06 R↑	These steps alternately roll the stack up and copy the current contents of the X-register into a register in the file STACK. The number t (that started in the T-register) is copied first, followed by z, y, and x. The pointer advances automatically, placing t in 000, z in 001, y in 002, and x in 003. This order simplifies the third routine, which restores these values to the stack.
07 SAVEX	
08 R↑	
09 SAVEX	
10 R↑	
11 SAVEX	
12 R↑	
13 SAVEX	

The third routine copies the numbers back into the stack; it is executed after setting the alarm.

01 ^T STACK	Places the file name in the Alpha register.
02 CLX	Places zero in the X-register.
03 SEEKPTA	Makes STACK the current file and makes register 000 the current register.

04 GETX	Recalls *t* to the X-register, lifts the stack, and advances the pointer.
05 GETX	Recalls *z* to the X-register, lifts the stack, and advances the pointer.
06 GETX	Recalls *y* to the X-register, lifts the stack, and advances the pointer.
07 GETX	Recalls *x* to the X-register, lifts the stack, and advances the pointer.

Text File Operations

Most of the operations covered here move data between a text file and the Alpha register. Any information in the Alpha register—characters from the Alpha keyboard or bytes created by XTOA—can be moved to and from a text file by these operations.

For simple operations it is easier to use the Text Editor (ED) described in the next section. If you want to review a text file or edit it using standard characters, ED lets you see what you're doing while you're doing it. However, any program that *automatically* manipulates text files must use the operations covered here.

Checking Available Bytes

Before you add data to an text file you can check how many bytes are available, using ASROOM (*ASCII room*). The text file must be the current file. ASROOM returns the number of available bytes to the X-register, lifting the stack unless stack lift is disabled. Remember that each new record requires one extra byte for overhead. If there is not enough room, refer to RESZFL on page 213.

Record Operations

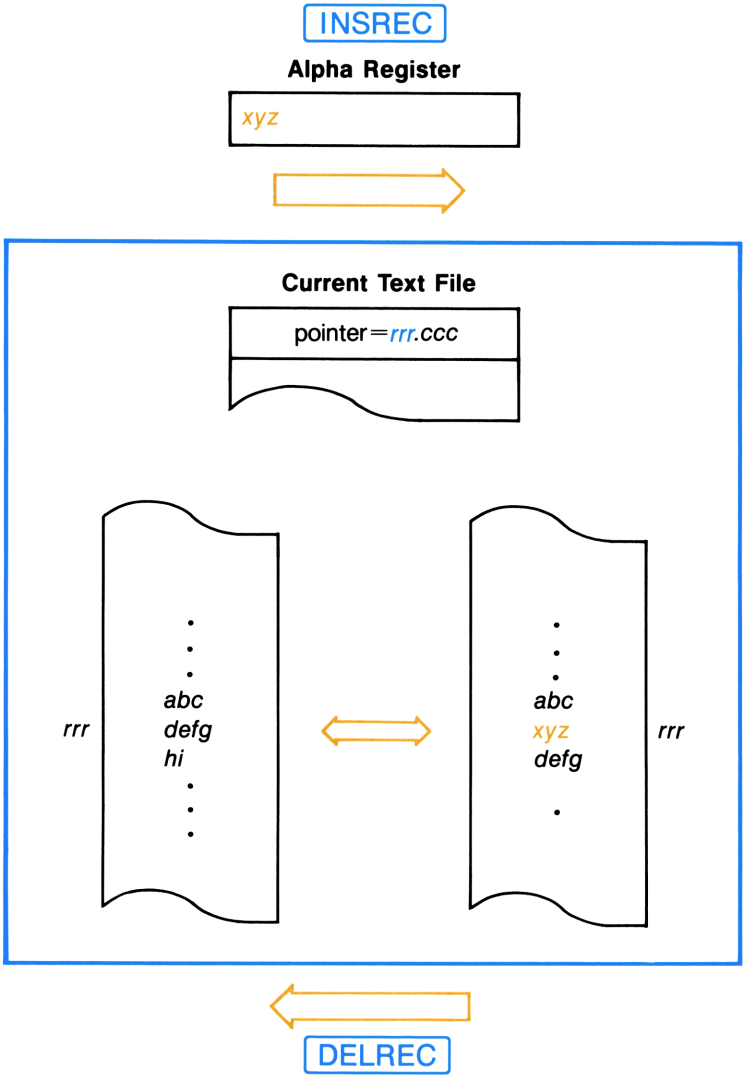
You can append, insert, or delete a record in a text file by using APPREC (*append record*), INSREC (*insert record*), or DELREC (*delete record*). The desired file must already be the current file. Both APPREC and INSREC make the contents of the Alpha register a new record in that file and advance the pointer just past the last character in the new record. This sets the pointer properly for adding characters to the new record.

Appending a Record. Use APPREC to append a new record:

1. Be sure that the desired text file is the current file.
2. Place the data for the new record in the Alpha register.
3. Execute APPREC.

Inserting a Record. Use INSREC to insert a new record:

1. Be sure that the desired text file is the current file and that the record pointer is set to the desired position of the new record. (The value of the character pointer doesn't matter.)
2. Place the data for the new record in the Alpha register.
3. Execute INSREC.



Deleting a Record. Use DELREC to delete a record:

1. Be sure that the desired text file and record are the current file and record. (The value of the character pointer doesn't matter.)
2. Execute DELREC.

The record pointer value is unchanged, but the current record is now the record that previously followed the deleted record. The character pointer is set to 000.

Character Operations

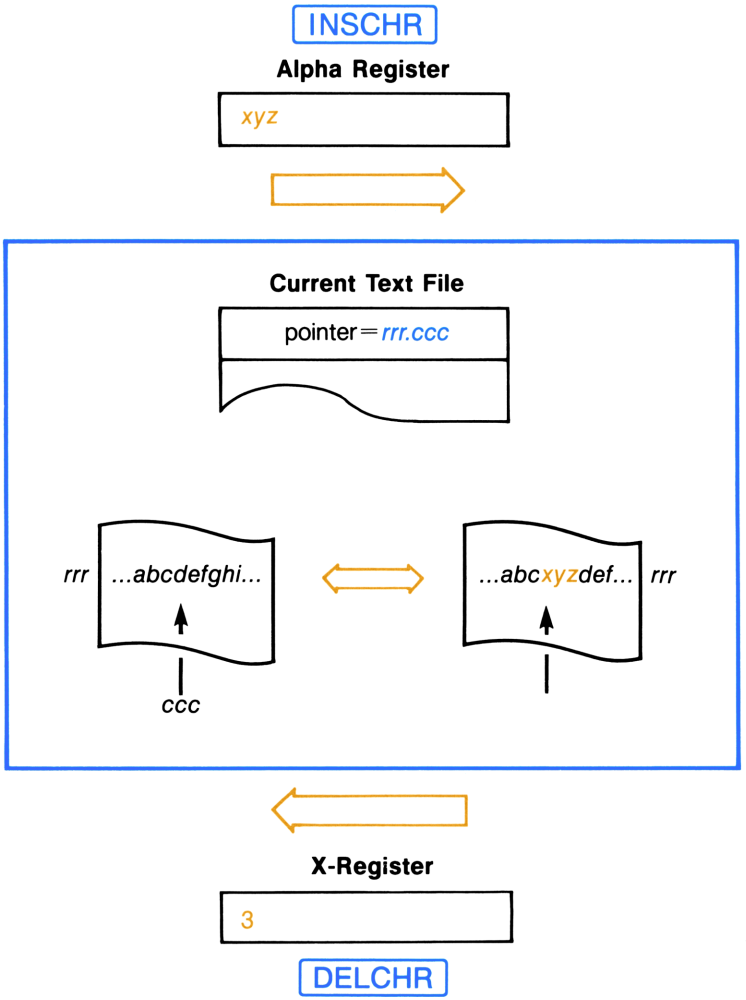
You can append, insert, or delete characters in a record by using APPCHR (*append characters*), INSCHR (*insert characters*), or DELCHR (*delete characters*). The desired record must already be the current record in the current text file. Both APPCHR and INSCHR add the contents of the Alpha register to the record and advance the pointer just past the last character added. This allows you to repeat either function immediately after placing new data in the Alpha register.

Appending Characters. Use APPCHR to append characters to a record:

1. Be sure that the desired text file and record are the current file and record. (The value of the character pointer doesn't matter.)
2. Place the characters to be appended into the Alpha register.
3. Execute APPCHR.

Inserting Characters. Use INSCHR to insert characters in a record:

1. Be sure that the desired text file and record are the current file and record, and that the character pointer is set to the desired position for insertion.
2. Place the characters to be inserted into the Alpha register.
3. Execute INSCHR.



Deleting Characters. Use DELCHR to delete characters in a record:

1. Be sure that the desired text file and record are the current file and record, and that the first character to be deleted is the current character.
2. Place in the X-register the number of characters to be deleted.
3. Execute DELCHR.

The character pointer value is unchanged, but the current character is now the character that previously followed the last deleted character.

Searching a File

You can search a text file for a particular string of characters by using POSFL (*position in file*) as follows:

1. Make sure that the file to be searched is the current file.
2. Set the pointer to the position where you want the search to begin.
3. Place the target string in the Alpha register.
4. Execute POSFL.

If the target string is found:

- The pointer is set to the first character of the string.
- This pointer value is returned to the X-register.

If the target string is *not* found:

- The text pointer is unchanged.
- A value of −1 is returned to the X-register.

In either case the Alpha register is unaltered, and the stack lifts unless stack lift is disabled. If the target string was found, you can recall that part of the file to the Alpha register using the following functions.

Copying Data into the Alpha Register

GETREC (*get record*) and ARCLREC (*Alpha recall record*) copy characters from a text record into the Alpha register. They differ only in whether they first clear the Alpha register.

- GETREC clears the Alpha register before copying characters.
- ARCLREC appends the copied characters to the previous contents of the Alpha register.

Otherwise, the two functions follow the same procedures.

- Both start with the current character in the current record in the current file.
- Both copy sequential characters until the last character in the record is copied or until the Alpha register is full. (This means that ARCLREC does nothing if the Alpha register is already full.)
- Both clear flag 17 if the last character in the record is copied or set set flag 17 if not. (For certain HP-IL peripherals, this flag controls whether sequential outputs are considered separate lines.)
- Both advance the character pointer just past the last character copied.

To execute GETREC or ARCLREC, make sure that the desired file is the current file and that the pointer is set as intended, and then execute the function. After you have processed the recalled data in the Alpha register, you can test flag 17 to check whether there are more characters to be copied in that record.

Accessing Mass Storage Files

You can make permanent copies of text files with a mass-storage device controlled by an HP 82160A HP-IL Module. SAVEAS (*save ASCII*) copies a text file from extended memory to mass storage, and GETAS (*get ASCII*) copies the file from mass storage back to extended memory. The following rules apply to saving and recalling files:

- The destination file must exist before the source file can be copied to it. To create the destination file on the mass storage medium, use CREATE (a function in the HP-IL module) to create a data file. It will become a text file when you execute SAVEAS.
- If the destination file is smaller than the source file, as much data as possible is copied before an **END OF FL** error stops program execution. You can set flag 25 (Error Ignore) if you intend to make an incomplete copy of the source file.
- Specify the source and destination files, separated by a comma, in the Alpha register.

Alpha Register

source-file name,destination-file name

If the two files have the same name, you can place just that name in the Alpha register.

Section 14

The Text Editor

Contents

Introduction

This section tells you how to use the Text Editor, a tool for working with text (ASCII) files. The discussion here assumes that you are familiar with the creation, deletion, and memory requirements of text files as described in section 13.

Adjusting the record/character pointer was integral to the use of the text-file operations (like APPREC, APPCHR, and INSCHR) described in section 13. This is because these operations use the Alpha register as an intermediary, passing up to 24 characters between you and the text file. *They do not allow you to read the text file itself.*

The Text Editor lets you view text files directly. It provides the means to directly type in, edit, and view the contents of text files. The Text Editor greatly simplifies the writing and modification of text files. However, there are two limitations to the Text Editor: its operations cannot be programmed (while functions like DELCHR can be), and you cannot use it to enter characters not on the Alpha keyboard (which you can with APPCHR or INSCHR).*†

The Text Editor's complete operation is described in section 8, "Storing Text, Data, and Programs in Files." Therefore, this section is organized for easy reference, with very short explanations. There is also an entry for ED in the Function Table "Extended Memory Functions," on page 426.

The Text Editor Display

The Window

When you execute ED, the display becomes a 12-character window into a record within the specified text file. The cursor (underscore) blinks alternately with a character or blank; the position of the cursor and the record shown are determined by the current value of the record/character pointer.

The Record Number. For convenience, the left end of a record display shows a two- or three-digit record number. This is not part of the record itself.

The Empty-Record Indicator (ᵀ). An empty record is shown by a record number followed by the "raised T" empty-record indicator, ᵀ.‡ This symbol appears in a newly created record, as well as in a record that loses its last character.

Punctuation. A punctuation character (".", ",", or ":") appears to the right of a regular character position. Therefore, a punctuation mark does not count as one of the 12 character spaces, unless there are two or more punctuation marks in a row. (Adjacent punctuation marks are separated by the width of a character.) Since the cursor is a full character (the underscore), it cannot occupy the same position as a punctuation mark—so when the record/character pointer is at a punctuation mark, the punctuation mark just blinks, with no cursor.

The Cursor. Except when the display shows the beginning or end of a record, the display window surrounds the cursor and the current character in the approximate center of the window. The cursor can be moved right as far as one character position past the end of the record. This is its position when you are adding characters to the end of a record.

---

* ED, which activates the Text Editor, is programmable, but none of its operations are. See "Using ED in a Program," page 233.

† To enter a nonstandard character (like @) in a text file, you need to enter it first into the Alpha register by using XTOA and then enter it into the text file by using APPCHR or INSCHR.

‡ The empty-record indicator is actually a "dummy character": it is like a character except that it can't be deleted when it is marking an otherwise empty record. The ᵀ uses one byte of memory (like any other character), so an empty record is *not strictly* empty. (The function ASROOM will count two bytes for each empty record: one for overhead and one for the empty-record indicator.)

Annunciators

The annunciators appearing in the display are redefined for the Text Editor:

- **1** indicates that Insert mode is active (page 232).
- **ALPHA** indicates that the regular Alpha keyboard (with the Text Editor control keys) is active; no **ALPHA** means the numeric keypad is active (page 232).
- **SHIFT** and **BAT** have their usual meanings.
- All other annunciators remain off, regardless of their status upon entering the Editor. Their prior status is restored upon leaving (exiting) the Editor.

Text Editor ([ED]) Operations

Entering and Exiting the Text Editor

To access a text file with the Text Editor:

1. If it doesn't already exist, create a file of a specific name and size using [CRFLAS]. (See page 212.)
2. With the name of the desired file in the Alpha register, execute [ED] (*editor*). An empty Alpha register specifies the current file.
3. The display will show the cursor on the current character in the current record (determined by the record/character pointer). A new file starts with one empty record (numbered 000).

To exit the Text Editor, press the [ON] toggle key ([EXIT] on the [ED] keyboard). An *automatic* exit occurs if you try to put more than 254 characters (the limit) in a record, or after a few minutes of inactivity. Exiting restores all annunciators that were redefined by [ED].

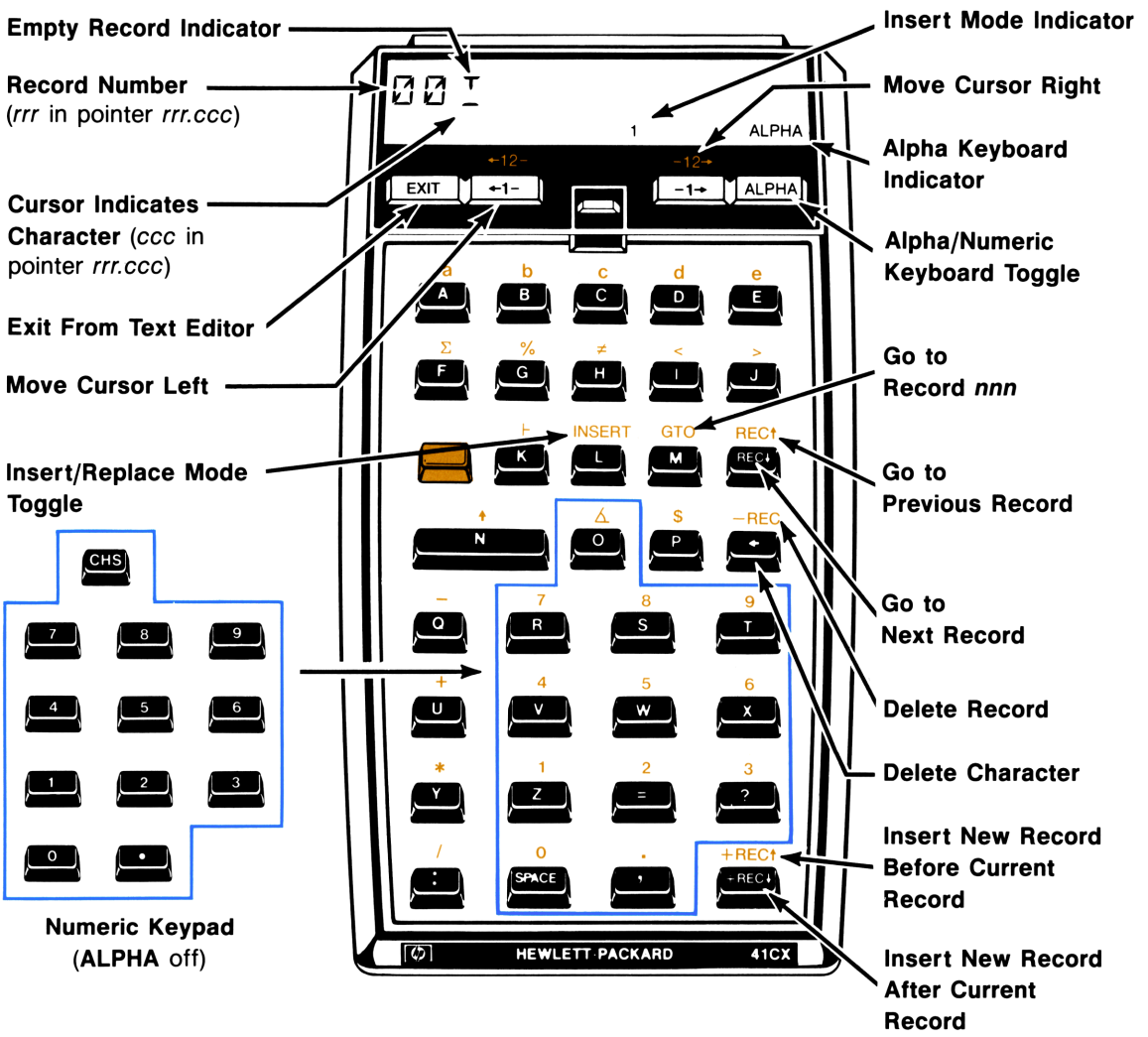
Default Conditions

Each time you activate the Text Editor manually, the Alpha (rather than Numeric) keyboard is active (**ALPHA** on), and Replace mode is active (new characters write over old ones). If [ED] is executed from within a program, the status of the Alpha flag (48) remains as it was before [ED] was executed so that the program can control which keyboard is active.

The Keyboard

An annotated diagram of the Text Editor ([ED]) keyboard is on page 231. The backplate of the HP-41CX has a diagram of the Text Editor keyboard, and the Quick Reference Guide also contains one. This keyboard is composed of two basic types of keys: the Alpha character keys (including digits), and the control keys, which supply the editing functions.

The Text Editor Keyboard



The character set on the Text Editor keyboard matches the Alpha character set, including the shifted characters and digits. This is the default character set when you execute [ED] manually.

The Numeric Keypad

Using the [ALPHA] toggle key to extinguish the Alpha annunciator will set the numeric keypad, shown as an insert on the keyboard diagram. The numeric keypad is for convenience when entering numbers: it makes the digit keys, [.], and [CHS] the primary, unshifted keys.* *All other character keys become inactive.* Upon exiting the Text Editor, the status of the Alpha annunciator (flag 48) will be restored to what it was before you entered the Editor.

The Character Control Keys

The annotation on the keyboard diagram (page 231) indicates all of the control keys.

Cursor Control. [←1] ([USER]), [1→] ([PRGM]), [←12] ([■] [USER]), and [12→] ([■] [PRGM]). The cursor represents the record/character pointer. You cannot move the cursor beyond the beginning of the record; you cannot move the cursor any further than one character position past the end of the record.

The sequence [REC↓] [REC↑] or [REC↑] [REC↓] will instantly move the cursor to the beginning of a long record.

Character Addition. [INSERT] ([LBL]) toggles between Replace (character) and Insert (character) modes. The default condition is Replace; no annunciator lit. Insert mode is signified by the **1** annunciator.

In Replace mode (the default mode), any character you key in writes over the character indicated by the cursor. In Insert mode, a character typed in is inserted ahead of the cursor.

Character Deletion. [←] deletes the character indicated by the cursor; the characters on the right shift left to fill the gap. If the cursor is past the last character in the record, pressing [←] deletes the last character. If there is only one character left and it is deleted, the T (the empty-record indicator) appears.

The Record Control Keys

Record Addition. [+REC↑] ([VIEW]) and [+REC↓] ([R/S]) insert an empty record before or after the current record. The T empty-record indicator is displayed. ([+REC↑] and [+REC↓] also terminate Insert mode if it was active.)

* The [.] key will record either "." or "," depending on the current radix mark convention (flag 28).

Record Deletion. [−REC] ([CLx]) deletes the current record. The cursor (the record/character pointer) moves to the first character of the next record unless the deleted record was the last one, in which case the cursor moves to the first character of the previous (now the last) record.

Moving Among Records. [REC↓] ([SST]) and [REC↑] ([BST]) move the cursor (the pointer) to the first character of the next or previous record, respectively. The cursor will not move beyond the first or last record (the display just blinks if you try).

[GTO] *nnn* moves the cursor to the first character of the indicated record, which must be specified by three digits. Like the usual [GTO], this is a parameter function that cues you for a three-digit input.

To locate the last record quickly, use a very large number with [GTO], such as [GTO] 999.

Errors

NO ROOM (with a beep) occurs when there is no more memory space in the file to add more characters or records. This is just a warning message, lasting about 1 second. There is no other effect. If you need more room in the file, allocate more registers using [RESZFL], page 213.

REC TOO LONG occurs if you try to exceed the maximum record length (254 characters). This causes the HP-41 to exit the Text Editor. (Just re-execute [ED] to recover your place.)

FL NOT FOUND occurs if you execute [ED] when the contents of the Alpha register do not correspond to the name of an existing file.

NAME ERR occurs if you execute [ED] when the Alpha register contains an illegal name, such as " " or seven bytes of decimal value 255 (▮▮▮▮▮▮▮).

FL TYPE ERR occurs if you execute [ED] when the contents of the Alpha register specify a non-text file.

Using [ED] in a Program

As mentioned in the beginning of this section, [ED] is programmable, though its control operations are not. Therefore, to use [ED] in a program there must be a user present to respond and enter or edit the necessary text when the program executes [ED].

```
01 "SECRETS
02 ED
```

All you *need* to do in the program is put the name of the desired file in the Alpha register and execute [ED]. Program execution will automatically stop, showing the Text Editor display. (If there is no response within a couple of minutes, the Text Editor will time-out—that is, be cancelled, and the program will continue running.)

```
01 AON
02 "SECRETS
03 0
04 SEEKPTA
05 ED
06 AOFF
```

However, it is more complete to ensure as well that the Alpha keyboard is on ([AON]), if having it on is desirable, and that the cursor (alias record/character pointer) is at the desired location. Lines 06 and 07 set the record/character pointer to record 000, character 000. (Pointer operations are in section 13.)

A program that executes [ED] will stop to wait for input. When text entry or modification is complete, press [EXIT] to continue the program, *not* [R/S]. Remember that [R/S] means [+REC↓] to the Text Editor.

To make an automatic, programmed modification to a text file, you must use programmable functions like [INSCHR] and [APPREC]. (See section 13.)

Part IV
Time Functions in Detail

Section 15

Clock and Date Functions

Contents

Included within the HP-41CX is a crystal-based clock and a variety of time-based functions. The clock runs even when the computer is off. This section covers those time functions related to clock time and dates. Many of these functions are discussed with examples in section 6; the discussion in this section is structured for ease of reference and for completeness. These functions are also summarized in the Function Table "Time Functions." Many of the time functions are used in section 22, "Programs for Keeping Time Records."

The clock is maintained separately from the rest of the computer functions and Continuous Memory. Therefore, clearing Continuous Memory does *not* reset the time or stopwatch, though it *does* clear all alarms and the month/date format. The details are listed under "Effects of Clearing Memory, Power Interruption, and Low Power" in appendix F.

There are two important appendices that elaborate on aspects of the time functions: appendix B, "More About Past-Due Alarms," and appendix F, "Time Specifications." Appendix B is of interest if you set many alarms, and appendix F is of interest regarding the accuracy of the timer and the variables of power consumption.

Setting and Adjusting the Clock Time

1. Use [SETIME] to initially set the clock time. Use [SETDATE] to set the date.
2. Use [T+X] to account for errors made with [SETIME] and time zone changes. (If it is necessary, this will change the date, too.)
3. Wait at least a week, then use [CORRECT] to correct for time "drift" (inaccuracy).

If setting or adjusting the time or date causes any alarms in memory to be bypassed, then two tones sound and those alarms become past-due. Refer to "Past-Due Alarms," section 16.

Setting the Time With [SETIME]

The function [SETIME] sets the internal clock with a precision to hundredths of a second. While setting the hours, use the conventions shown in the table to the right.

To set the time:

1. Place the time in the form ±***HH.MMSSss*** into the X-register.
2. Execute [SETIME].

The time is set when you release the last key that executes [SETIME]. The maximum precision in manually setting the time is about 0.1 second. For greater precision, you can adjust the time using [T+X], below.

Time Settings

Clock Time	Setting
Midnight	0
1 (a.m.)	1
2	2
⋮	⋮
10	10
11	11
Noon	±12
1 p.m. or 13:00	−1 or 13
2 or 14	−2 or 14
⋮	⋮
10 or 22	−10 or 22
11 or 23	−11 or 23
Midnight	0

Adjusting the Time With T+X

The T+X (*time plus X*) function increments or decrements the current time according to the number in the X-register. Use T+X to alter the time due to local time changes (time zones, daylight savings) or inadequate *precision* in setting the time. Do *not* use T+X to accommodate *inaccuracy* in the timekeeping of the timer. To correct accumulated error in the time, use CORRECT, below.

To increment or decrement the time:

1. Place the time change (positive or negative) in the form ±***HHHH.MMSSss*** into the X-register.
2. Execute T+X.

If the time change specified moves the current time into a different day, T+X also changes the date. ("Setting the Date" is discussed later in this section.)

Adjusting the Accuracy With CORRECT

The CORRECT (*correct accuracy factor*) function both corrects the time setting and adjusts the accuracy factor. The accuracy factor compensates for time deviations due to normal variations in power, temperature, and manufacture, and continues to make those adjustments in the future. Therefore, CORRECT is not for correcting one-time anomalies in the time (see T+X, above). (In order to reduce the error introduced by keystroke imprecision, the time span between uses of CORRECT should be 1 week or more.)

The mechanics of this function are explained in detail in appendix F ("Time Specifications"), but the basic operation is:

1. Place the correct time in the form ***HH.MMSSss*** in the X-register.
2. Execute CORRECT at the appropriate instant. (The time you specify (step 1) will become the corrected time at the instant that CORRECT is executed.)

This sets the time and adjusts an internally calculated accuracy factor based on previous executions of SETIME, CORRECT, and other functions. The clock runs from the corrected time at a rate regulated by the newly calculated accuracy factor. (See appendix F.)

Displaying the Clock

Turning the Clock On (CLOCK and ■ ON)

- Pressing ■ ON or executing CLOCK will display the running clock. Only CLOCK is programmable.
- The sequence ■ ON turns the HP-41CX off before displaying the clock. This resets flags 12 through 20, as explained in section 12.
- Pressing ← clears the clock display and returns the display to the X-register. Pressing most other keys (including ON) clears the clock display and executes that function.

The exact composition of your clock display depends on the various clock-display formats, discussed under the next heading.

> **Note:** The clock display consumes a higher than usual amount of power. See "Power Consumption" and "Low Power" in appendix G.

As mentioned in section 19, the user flags 12 to 20 are reset each time the computer is turned on. Executing [ON] affects these flags in the same way as turning on the computer because the computer turns off momentarily before displaying the clock. However, [CLOCK] does *not* shut the HP-41 off, so if you want to avoid the resetting of flags 12 to 20, use [CLOCK]. For convenience, [CLOCK] can be assigned to a User key.

Clock Display Formats ([CLK24], [CLK12], [CLKT], [CLKTD])

You can choose the format of the clock display. The settings you choose are maintained even when Continuous Memory is reset, and until there is an interruption in power (like prolonged battery removal). The default display shows the time only, using a 12-hour clock (using a.m. and p.m.). These conventions can be changed as follows:

12- Versus 24-Hour Clock. Executing [CLK24] (*clock, 24-hour*) sets a 24-hour clock display. Executing [CLK12] (*clock, 12-hour*) sets a 12-hour clock display. Note that these functions do not affect how you must input clock-time values, nor how time values for time calculation functions are output. See the chart "Time Settings" on page 237. In other words, these functions affect the format of the clock display but not that of the X-register.

Time Only Versus Time and Date. Executing [CLKT] (*clock time*) sets the clock display to show the time only. Executing [CLKTD] (*clock time and date*) sets the clock display to show the time and the date. The exact display of the date depends on the month-and-day convention, as explained under "Date Formats," later in this section.

For instance, assuming a month/day format for 3:15 p.m. on a January 21st:

Clock Display Formats

Format	[CLKT]	[CLKTD]
[CLK12]	3:15:00 PM	3:15 PM 01/21
[CLK24]	15:15:00	15:15 01/21

The date display suppresses the seconds portion of the time display.

> **Note:** A time-date display ([CLKTD]) consumes less power than a time-only display ([CLKT]) because the time-date display updates itself only once a minute, while the time-only display updates itself every second.

Manipulating Time Values

A value representing the current time can be recalled to the X-register using [TIME]. A number already in the X-register can be converted to a time display format and appended to the Alpha register using [ATIME] or [ATIME24].

Recalling the Current Time Value ([TIME])

The function [TIME] places a number representing the current time into the X-register, lifting the stack (unless stack lift is disabled). The number is given in a 24-hour format, with six decimal places (hundredths of seconds): ***HH.MMSSss***. You need to set [FIX] 6 to see all six places. Midnight is zero.

In addition, if [TIME] is executed manually, you see a message display showing the time value according to the 12-hour or 24-hour format in effect (this is not the X-register). Pressing [←] clears the message display and displays the X-register.

At 5:05:24 p.m. ([FIX] 6), executing [TIME] manually, followed by [←], results in:

Time Values

Format	[TIME] (temp. display)	[←] (X-register)
[CLK12]	5:05:24 PM	17.052400
[CLK24]	17:05:24	17.052400

Appending a Number in Time Format to the Alpha Register ([ATIME])

The [ATIME] (*Alpha time*) function appends the number in the X-register to the contents of the Alpha register in the current clock display format: [CLK12] or [CLK24]. This function is useful for programs involving the time of day.

To indicate a value for elapsed time, use [ATIME] with [CLK24], or use [ATIME24] (the following topic). In these cases, there will be no **AM** or **PM** or alteration of numbers from 13 to 23.

The integer (hours) part of the number in the X-register determines how the number is formatted in the Alpha register:

- [ATIME] accepts any number in the range $-100 < x < 100$. The negative sign is ignored *except* for -1 through -11, which are interpreted as post-meridian times, as indicated in the chart "Time Settings" on page 237.
- If $|x| \leq 23$, the number is formatted according to the current [CLK12] or [CLK24] format setting.
- If $|x| \geq 24$, no **AM** or **PM** are used. (A minus sign is ignored.) This is so you can output a value for an elapsed time, such as 30:12:00 (30 hours and 12 minutes).

For instance, assuming [FIX] 6:

Appending a Time to the Alpha Register

Value in X-Register	Appends to Alpha Register	
	in [CLK12]	in [CLK24]
−11	11:00:00.00 PM	23:00:00.00
−15.25	3:25:00.00 PM	15:25:00.00
30.125633	30:12:56.33	30:12:56.33

- The number appended to the Alpha register is truncated (not rounded) according to the current numeric display setting:

[FIX], [SCI], [ENG]	Appends to Alpha Register	Example: 10.0637 (10:06:37 AM in [CLK12])
0	*HH*	10 AM
1 or 2	*HH:MM*	10:06 AM
3 or 4	*HH:MMSS*	10:06:37 AM
5 or more	*HH:MMSSss*	10:06:37.00 AM

If there is not enough space left in the Alpha register to accommodate its existing contents plus the characters appended by [ATIME], the leftmost characters in the Alpha register are lost to make room for the new characters appended on the right.

If the X-register contains a number outside the range $-100 < x < 100$, executing [ATIME] causes a **DATA ERROR**.

Appending a 24-Hour Time Value to the Alpha Register ([ATIME24])

The function [ATIME24] (*Alpha time, 24-hour*) operates like [ATIME] except that the number appended to the Alpha register is always expressed either in 24-hour format or as elapsed time. The current clock display format has no effect.

Setting and Manipulating the Date

Functions included in this topic are date formatting, setting the date, recalling the date, and appending a number in date form to the Alpha register.

Date Formats (MDY, DMY)

The default setting for the date format is month/day/year. The formatting functions MDY (*month/day/year*) and DMY (*day/month/year*) set the date format to the month/day or the day/month convention. This setting is the only time format to be reset when Continuous Memory is reset.

The following table shows how the computer interprets numeric input as dates and how date output is formatted.

Effect of Date Formats

Format Setting	X-Register Input/Output	Display and Printer Output	Flag 31
MDY	*MM.DDYYYY*	*MM/DD* or *MM/DD/YY*	Clear
DMY	*DD.MMYYYY*	*DD.MM* or *DD.MM.YY*	Set

When entering dates you can omit leading and trailing zeros. For example, a number representing May 6, 1990 can be entered as 5.06199. The appropriate leading and trailing zeros do appear in message-type displays of the date.

Setting the Date (SETDATE)

To set the date:

1. Put the date in the X-register according to the current date format (see above), using the form ***MM.DDYYYY*** or ***DD.MMYYYY***. As mentioned above, you can omit leading and trailing zeros.
2. Execute SETDATE.

The HP-41 can be set to any date from January 1, 1900 to December 31, 2199.

If the format of the date you enter does not correspond to the current month-and-day setting, no error message will result unless the date interpreted from your input is actually an invalid date. If setting the date causes any alarms to become past-due, two tones will sound. Refer to "Past-Due Alarms" in section 16.

Recalling a Current Date Value (DATE)

The function DATE returns a number representing the current date to the X-register, lifting the stack (unless stack lift is disabled).

- The number recalled to the X-register will be in the form ***MM.DDYYYY*** or ***DD.MMYYYY***, depending on the date format setting.
- In addition, if DATE is executed manually, a message display shows the date and day of the week in the form ***MM/DD/YY DAY*** or ***DD.MM.YY DAY***, where ***DAY*** is a three-letter abbreviation for the day of the week. (For the years 2000 to 2199, the date display is ***MM/DD/YYYY:DA*** or ***DD.MM.YYYY DAY***.)

Press [←] to return the X-register to the display.

Appending a Number in Date Format to the Alpha Register ([ADATE])

[ADATE] (*Alpha date*) appends the number in the X-register in date format to the current contents of the Alpha register.

- The input should be of the form ***MM.DDYYYY*** or ***DD.MMYYYY***, depending on the date format. The valid input range is $|x| < 100$, so that you can represent periods of elapsed calendar time up to 99.999999 (months or days).
- The appended number is converted to the form ***MM/DD/YYYY*** or ***DD.MM.YYYY***. (A minus sign is ignored.) Leading and trailing zeros are shown.
- The appended number is truncated (not rounded) according to the current numeric display setting:

Appending a Date to the Alpha Register

[FIX], [SCI], [ENG]	Appends to Alpha Register		Example: 6.11196 (June 11, 1960 in [MDY])
0	*MM*	or *DD*	06
1 or 2	*MM/DD*	or *DD.MM*	06/11
3 or 4	*MM/DD/YY*	or *DD.MM.YY*	06/11/60
5 or more	*MM/DD/YYYY*	or *DD.MM.YYYY*	06/11/1960

Notice that when only two year digits appear, they are the last two year digits.

Calculations With Dates

There are three functions that perform calculations with dates.

- [DATE+] adds or subtracts a number of days from a given date and determines the resulting date.
- [DDAYS] calculates the number of days between two given dates.
- [DOW] calculates the day of the week for a given date.

Valid Dates

- The only valid input and output dates for these calculations are from October 15, 1582 (the beginning of the Gregorian calendar) through September 10, 4320.
- The input (***MM.DDYYYY*** or ***DD.MMYYYY***) *must* be positive. If there are any digits after the ***YYYY***, they must be zeros.

The errors **DATA ERROR** and **DATA ERROR Y** can result if data input is invalid. An invalid result for [DATE+] will cause **OUT OF RANGE**.

Date Arithmetic (DATE+)

DATE+ (*date plus*) calculates a new date given the old date (in the Y-register) and the number of days to add or subtract (in the X-register).

1. Enter the known date (***MM.DDYYYY*** or ***DD.MMYYYY***).
2. Enter the number of days to add to or subtract from that date. Use a positive number for addition, a negative number for subtraction. Only the integer part of this number is used.
3. Execute DATE+. The stack drops, placing the resulting date in the X-register in the form ***MM.DDYYYY*** or ***DD.MMYYYY***. (The previous contents of X are saved in the LAST X register.)

The Number of Days Between Dates (DDAYS)

Given a date in the X-register and a date in the Y-register, DDAYS (*delta days*) finds the difference, in days, between those two dates. The date in Y is subtracted from the date in X.

1. Enter the first (earlier) date (***MM.DDYYYY*** or ***DD.MMYYYY***), then the second one. (If the first date entered is the later one, the result will be negative.)
2. Execute DDAYS. The stack drops, placing the number of days of difference into the X-register. (The second date is saved in the LAST X register.)

Day of the Week (DOW)

The function DOW (*day of week*) converts the date in the X-register (***MM.DDYYYY*** or ***DD.MMYYYY***) into a value for its day of the week.

1. Enter the date.
2. Execute DOW.
3. The value in the X-register is converted to a number from 0 (Sunday) to 6 (Saturday). (The date is saved in the LAST X register.)

If this function is executed manually, a message display appears with a three-letter abbreviation for the day of the week. To see the X-register (with the numeric day-of-week value), press ←.

Limits and Errors

Limits

All time and date functions except [shift] [ON] are programmable.

Clock Time: Minute-values (***MM***) and second-values (***SS***) are valid from 00 to 59 only. (Values for hundredths of a second (***ss***) run from 00 to 99, as usual.)

Dates: The date setting can be any date from January 1, 1900 (the default date) to December 31, 2199.

Calendar Functions: For *calculations* involving dates, any date from October 15, 1582 (the beginning of the Gregorian calendar) through September 10, 4320 is valid.

Display: Since the time is kept to hundredths of a second, a [FIX] 6 display format is necessary to see the full result of some functions.

Errors

DATA ERROR occurs if the *input* for a time function is not in the proper format or is not within the valid range. ([ADATE], [ATIME], [ATIME24], [CORRECT], [DOW], [SETAF], [SETDATE], [SETIME], [SETSW], [T+X])

The error **OUT OF RANGE** occurs if the result of a time function would fall outside the valid range. ([T+X], [DATE+])

Section 16

Alarm Functions

Contents

Alarm operations in the HP-41CX are quite versatile: depending on the parameters you supply, an alarm can be simply a personal reminder (an audible alarm with a message), or it can be a sophisticated control device to start the execution of a program at a certain time. In any case, an alarm will not interrupt the execution of a function in progress, but waits until that one function (including a function being executed by a program) is finished before going off.*

There are three distinct types of HP-41 alarms, but they are all set in a similar manner. Once set, alarms are stored in main memory along with programs and key assignments. Section 6, "Time Functions", offers an introduction to alarms, with examples. These functions are summarized in the Function Table "Time Functions".

A program executed by an alarm can affect the status of the flags or the various registers, as any other program can. This means you might want to make provisions in the called program to restore affected data you might need for other calculations.

Types of Alarms: Message, Control (↑↑), Conditional (↑)

The three different alarm types and their operation are outlined in the flowchart on page 249. Of these three alarms, one is used like an alarm clock (the message alarm) and two are used to execute programs (the control alarm and the conditional alarm, the latter being a "non-interrupting" control alarm).

Message Alarms

A message alarm is handy as an appointment reminder. This type of alarm produces a series of tones and flashes the message placed in the Alpha register when the alarm was set. The message can be up to 24 characters long. If the alarm is set with an empty Alpha register, a display of the current time and date flashes instead.

This kind of alarm goes off whenever its time arrives—whether the computer is on or off or executing a program. Remember, however, that no alarm interrupts the function currently in progress.

A message alarm needs to be acknowledged, otherwise it becomes past due (retained in memory as an unacknowledged alarm). (Past-due alarms are discussed fully in this section under "Past-Due Alarms" and in appendix B.) A message alarm replaces any previous message in the display, but otherwise has no effect on the computer's operation or status, including the stack registers, the data storage registers, and the Alpha register.

* This includes the very "long" functions of indefinite duration: running catalogs, catalogs 4 to 6 at all times, the Text Editor ([ED]), and the stopwatch ([SW]). Alarms will not activate during these operations. but will wait and go off afterwards.

Control Alarms (↑↑)

A control alarm executes the program or the programmable catalog-2 function specified by ↑↑ *global label* or ↑↑ *catalog-2 function name* in the Alpha register when the alarm is set.* It will execute a program starting at a global label, or, if no label or name follows the two up-arrows, at the current program line. When the set time arrives, the specified program or function is executed, *whether the HP-41 is on or off*, as soon as any currently operating function is done.

If a control alarm comes due while a program—including a program started by another control alarm—is running, that currently running program will be *temporarily* suspended while the program or catalog-2 function referenced by the interrupting, control alarm is executed.† The computer runs the called program as a subroutine of the interrupted one, using one subroutine level. (Subroutines and subroutine levels are explained in section 20, "Branching".)

A control alarm does not get acknowledged; it simply activates and then clears or resets itself.

Conditional Alarms (↑)

A conditional alarm, like a control alarm, will execute the program or programmable catalog-2 function specified in the Alpha register. However, it is a *non-interrupting control alarm* in that it will *only* take effect if the computer is *off* or the clock is displayed.

- If a program is running when a conditional alarm comes due, the alarm sounds a pair of tones and becomes past due.
- If the computer is on (no program running, no clock displayed) when the alarm comes due, the alarm becomes a message alarm, and does not execute the specified program or function. (The alarm becomes past due if it is not acknowledged, just like a message alarm.)

Therefore, whether a conditional alarm acts like a control alarm is dependent upon the computer being off. This is useful if you want a program to be automatically executed anytime *except* when you're in the process of performing other calculations or programs on the HP-41. A conditional alarm that acts as a control alarm does not get acknowledged; a conditional alarm that becomes a message alarm needs acknowledgment.

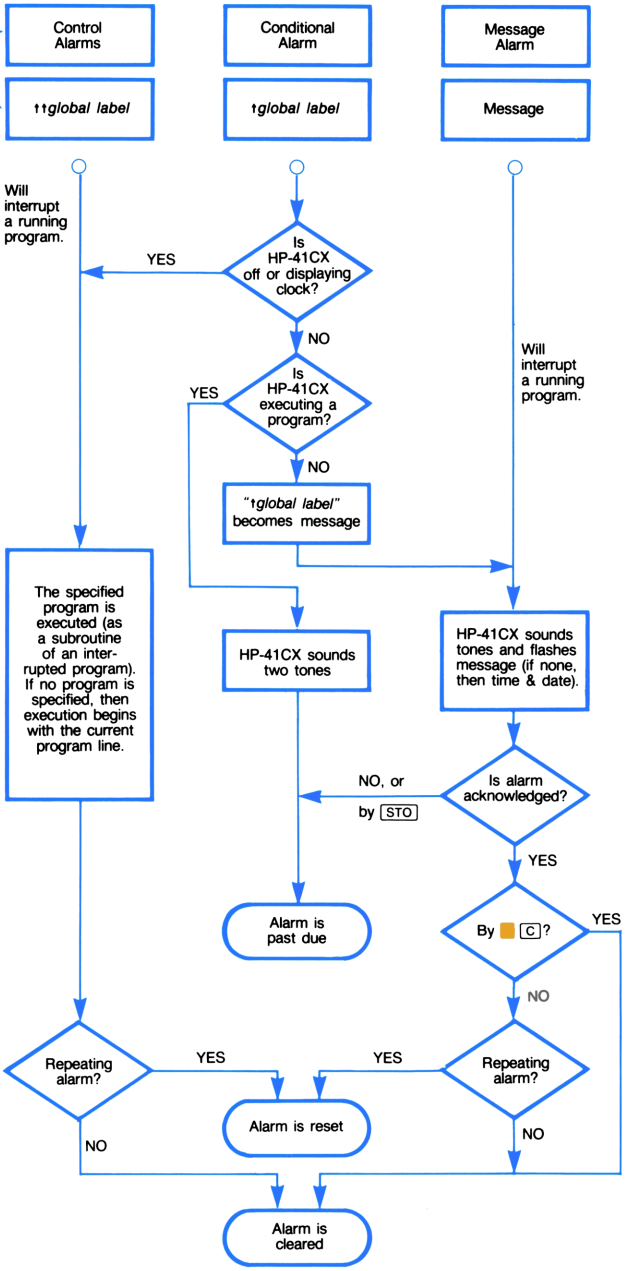
---

* For a description of catalog-2 functions, refer to "Cataloguing the New Functions" in appendix I.

† Program interruption occurs following execution of whatever function is in the process of being executed when the alarm comes due. Also, the alarm will not interrupt until stack lift is enabled.

Alarm Flowchart

The **alarm type** is determined by the presence in the **Alpha register** of two, one, or no up-arrows (↑) when XYZALM is executed. This chart shows what each type does when it comes due.



Below are comments on possible outcomes:

An alarm that has come due is either **cleared** from memory, **reset** according to its repeat interval, or becomes **past due**.

A **control alarm** will always be **cleared** (or **reset** if it is a repeating alarm). It will *never* become **past due**.

A repeating alarm will be **reset** only if it causes program execution or is acknowleged. An alarm which is **past due** will *not* be **reset**.

If a program was interrupted for an alarm, the interrupted program now resumes.

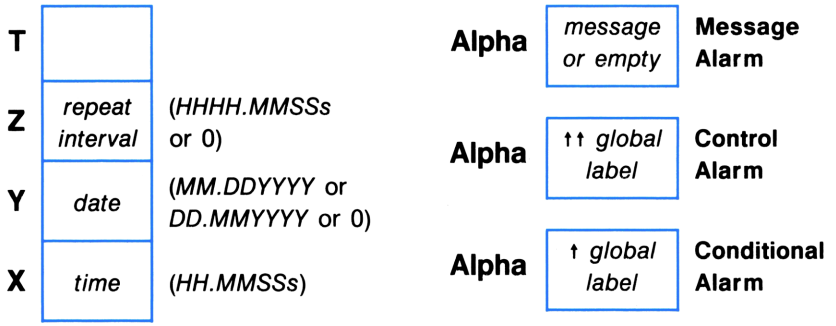
Setting Alarms (XYZALM)

The single, programmable function XYZALM (*X, Y, Z alarm*) is used to set all three types of alarms. The XYZALM function uses the data present in the X-, Y-, and Z-registers and the Alpha register. Time values can be as precise as tenths of a second.

1. Load the X-, Y-, Z- and Alpha registers with the required parameters (see below). *The contents of the Alpha register will determine whether the alarm is of type message, control, or conditional.*
2. Execute XYZALM.

An illegal input results in a **DATA ERROR** message, specifying the register with the faulty data (**DATA ERROR X**, etc.). A pair of tones sounds if the alarm is set to a past time, or if any bypassed past-due alarms are in memory.

Parameters for XYZALM



Z-Register: Alarm Repeat Interval. The interval after which the alarm will repeat itself. 1 second ⩽ interval < 10,000 hours. A *repeating alarm*—an alarm whose repeat interval is not zero—resets its future activation time according to the repeat interval and the original activation time (as also explained under "Activation and Acknowledgment of Message Alarms", page 253).

For no repetition, the Z-register must contain zero.

> **Note:** A repeating control alarm of very short repeat interval (less than 10 seconds) can be difficult to cancel. See "Clearing Alarms from Memory," page 258.

Y-Register: Alarm Date. The date for the alarm to activate. Valid dates are from January 1, 1900 through December 31, 2199.

For the current date, use zero.

X-Register: Alarm Time. The time of day for the alarm to go off ("activate"), from 0 to ±23.59599 (one-tenth second before midnight). (Remember negative numbers from 1 to 11 mean p.m. See the table "Time Settings" in section 15.)

Alpha Register: Message Alarm. An Alpha string of up to 24 characters; an empty Alpha register will produce a time/date display with the alarm.

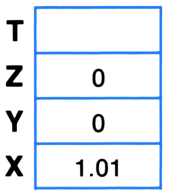
Alpha Register: Control Alarm. ↑ ↑ *global label* or ↑ ↑ *catalog-2 function name* or just ↑ ↑.* The label or name specified can have up to six characters (even though labels in program memory can have up to seven characters)—any more characters are simply ignored (even though the listing in the alarm catalog shows all characters).†

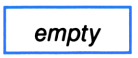
You can have a control alarm execute a program starting other than at a global label by putting *only* the two up-arrows (↑↑) in the Alpha register. Then, the alarm will trigger program execution *beginning at the current line in the current program* at the time the alarm comes due. This type of specification could be used, for instance, to set a program to resume execution at the same point it suspended itself with an OFF instruction.

Alpha Register: Conditional Alarm. ↑ *global label* or ↑ *catalog-2 function name*. The label or name specified can have up to seven characters.

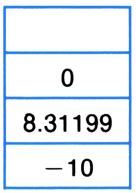
As with control alarms, you can set a conditional alarm to start program execution at the current line in the current program—rather than at a global label—by placing only the up-arrow (↑) in the Alpha register, without a label or function name.

Examples

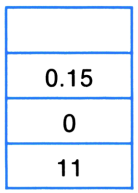




Executing XYZALM will set a nonrepeating message alarm for 1:01 a.m. on the current date. The "message" will consist of the time and date.



Will set a nonrepeating message alarm for 10:00 p.m. on August 31, 1990.

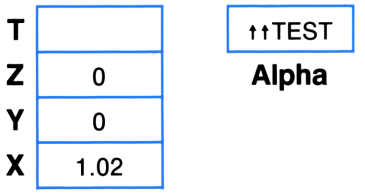


Will set a 15-minute repeating message alarm, starting at 11:00 a.m. on the current date.

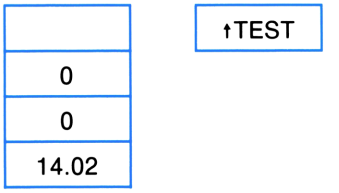
---

* The ↑ is on the Alpha keyboard above the N key (as shown on the backplate).

† To set a control alarm to execute a program or function that has a seven-character label or name, create a short "calling" program to execute the program or function you want, then set the alarm to execute the calling program.



Will set a nonrepeating control alarm for 1:02 a.m. on the current date. The alarm will call and execute program TEST. It will temporarily interrupt a running program, if necessary.

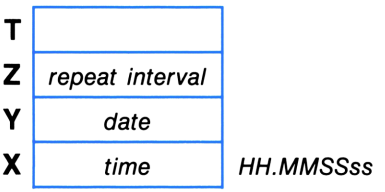


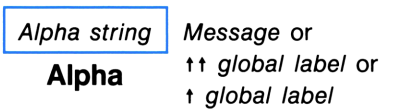
Will set a nonrepeating conditional alarm for 2:02 p.m. on the current date. The alarm will execute program TEST *only if* the HP-41 is off or displaying the clock.

Recalling Alarm Parameters to the Stack (RCLALM)

The function RCLALM (*recall alarm*) recalls the parameters of a stored alarm to the stack and Alpha registers, from which a program can examine and alter these values. Or, you can save the recalled values in other registers or in mass storage. If you wanted to delete most of your alarms but save a few, you could use RCLALM to save particular alarm values before using CLRALMS to delete all alarms ("General Clearing Operations", page 258).

RCLALM recalls to the X-, Y-, Z-, and Alpha registers the parameters of the alarm specified by the absolute value of the integer part of the number in the X-register. The number of the alarm corresponds to its chronological order as listed in the alarm catalog (see "The Alarm Catalog" page 255). Legitimate alarm numbers are from 1 to 253. A number greater than the number of existing alarms causes **NO SUCH ALM** (*no such alarm*).





The output format is the same as the input format for XYZALM, except that the time is always returned in a 24-hour format as ***HH.MMSSss***. The month/day format corresponds to the current setting. The contents of each register correspond to the register contents when XYZALM was executed for that alarm.

Executing RCLALM saves the alarm number (from the X-register) in the LAST X register, and lifts the previous contents of the Y-register into the T-register.

Activation and Acknowledgment of Message Alarms

Message alarms, *including conditional alarms that become message alarms*, follow a characteristic pattern of activation. They *must* be acknowledged (as described below) during the activation cycle, otherwise they become past due.

- Acknowledging a nonrepeating alarm shuts it off *and* clears it from memory.
- Acknowledging a repeating alarm shuts it off and resets its activation time to its next future occurrence. One or more multiples of the repeat interval are added to the *original alarm time*, not the time of acknowledgment.

Unacknowledged (past-due) message alarms are retained in memory: they are automatically reactivated when the computer is turned off, and set off warning tones whenever the computer is turned on. (Refer to "Past-Due Alarms.")

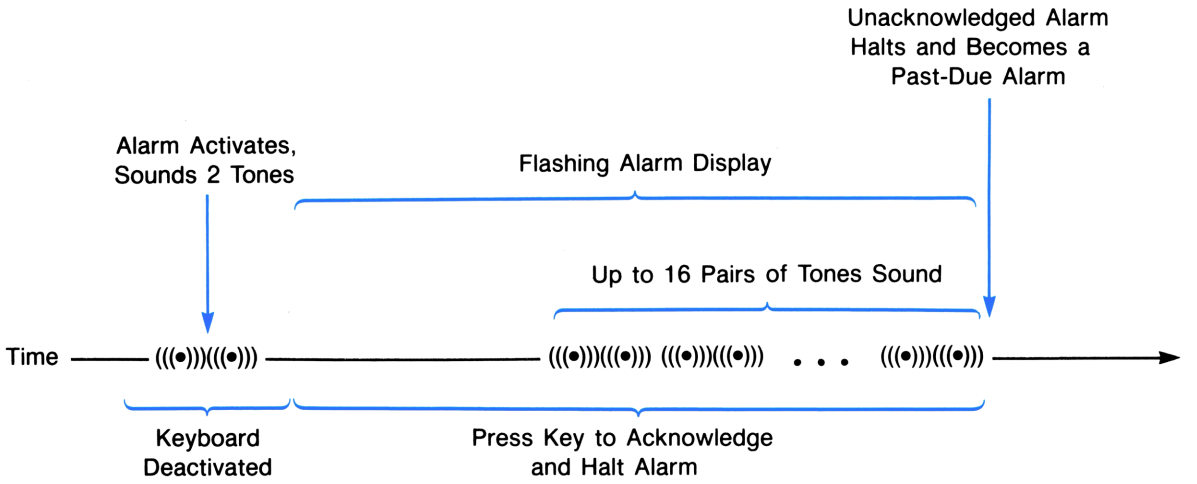
Control alarms, as well as conditional alarms that become control alarms, are self-acknowledging—they simply execute the program or catalog-2 function indicated, and then automatically clear themselves from memory (or reset themselves if they are repeating alarms).

The Activation Cycle

When a message alarm comes due, the following procedure starts:

1. It sounds a pair of tones and displays the first 12 characters of its Alpha string *or* the time and date (if no Alpha string was given for the alarm). (A conditional message alarm displays ↑ ***global label*** or ↑ ***function name***.)

   This phase lasts about 1 second, and the keyboard is inactive during this time.
2. The display begins flashing. *Starting now through step 3, you can acknowledge the alarm* (by pressing almost any key—refer to the following topic).
3. After the display flashes five times, the audible alarm starts again, sounding up to 16 pairs of tones.
4. If the alarm is not acknowledged by this time, it becomes past due. (Refer to "Past-Due Alarms.") If it is a repeating alarm, it is not reset.



When a message alarm is acknowledged, the above activation cycle stops. For a message of more than 12 characters, the ensuing display depends on the method of acknowledgment (below).

The Acknowledgment Procedure

While the display is flashing, and before the long series of tones is complete, you can acknowledge a message alarm by one of the following methods. Use these same procedures to acknowledge past-due alarms when they are automatically reactivated.

Results of Alarm Acknowledgment

Key Pressed		
← or ON	STO	Any Other Key
Halts alarm.	Halts alarm.	Halts alarm.
Immediately clears the alarm display.	Display persists for 3 seconds after release of key.*	Display persists for 3 seconds after release of key.*
	Pressing key again prolongs the display another 3 seconds.	Pressing key again prolongs the display another 3 seconds.
Clears the alarm.†	Retains the alarm as past-due.	Clears the alarm.†

* If there are more than 12 characters, the first 12 are shown while the key is held down, then the remaining characters are shown for 3 seconds.

† If it is a repeating alarm, it is reset and not cleared.

Be sure to wait until the alarm message has cleared before executing another function. Otherwise, you can delay the clearing of the alarm message.

Repeating Alarms. The new setting for a repeating alarm is determined by adding the repeat interval to the alarm time, not to the acknowledgment time.

If you use [STO] to acknowledge a repeating alarm, the alarm will not be reset—just retained as a past-due alarm. (The repeat interval is still maintained in the alarm catalog.) If a past-due repeating alarm is reset, it will not reset to a past time.

To clear from memory (delete) a repeating message alarm during its activation cycle, press [■] [C].

Delay of Activation. If an alarm comes due while an earlier message alarm is going off, the second alarm is delayed until the first one has been acknowledged or completes its activation cycle.

Simultaneous Alarms

If more than one alarm is set to exactly the same time, each alarm will activate in the order in which it was set. However, a control alarm will interrupt a program triggered by a control or conditional alarm. A message alarm will temporarily suspend the program executed by a previous control alarm (but, as noted in the previous paragraph, it waits for any other alarm activation cycles to finish).

If more than one simultaneous or overlapping alarm is a control alarm, successive alarms will interrupt the programs triggered by preceding alarms and execute their specified programs before the preceding programs can run. A conditional alarm will neither interrupt a program nor wait for completion; it simply sounds a pair of tones and becomes past due. Simultaneous alarms go off in the same sequence as past-due alarms, as explained in appendix B.

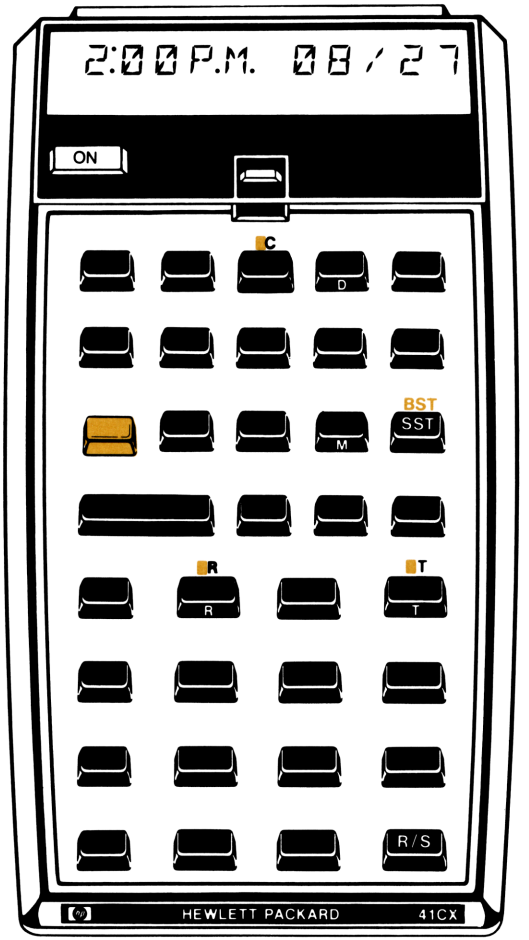
The Alarm Catalog ([ALMCAT], [CATALOG] 5)

The alarm catalog and the Alarm Catalog keyboard (the keyboard redefined for specific alarm catalog operations) are activated by either executing [ALMCAT] (*alarm catalog*) or pressing [CATALOG] 5. Only [ALMCAT] is programmable. For manual execution, [CATALOG] 5 is faster to execute.

The features of the alarm catalog are:

- It provides a list of all alarms currently kept by the HP-41 in memory (including past-due alarms).
- It lists the alarms in order of activation time, from earliest to latest. (The position of a repeating alarm is adjusted each time it is reset.)
- Each alarm listing shows first the time and date of the alarm, then any Alpha string (a message or ↑↑/↑ label call).
- When it finishes listing the last alarm, it exits the alarm catalog and returns to a display of the X-register.

The Active Keys on the Alarm Catalog Keyboard



- Its execution enables stack lift.
- Like other catalogs, it can be stopped and restarted with [R/S].* Once stopped, you can step through the catalog entries using [SST] or [BST] (the display just blinks when you hit the end/beginning of the catalog with [SST]/[BST]).
- While the alarm catalog is stopped, the keyboard is redefined as the Alarm Catalog keyboard.

The following keys are defined to perform the following operations on the Alarm Catalog keyboard while the alarm catalog listing is interrupted. These keys do not represent characters from the Alpha keyboard, so do not use [ALPHA]. Notice that the letter keys relate to the meaning of their Alarm Catalog function; for example, [T] for time.

The Active Keys on the Alarm Catalog Keyboard

Key(s)	Operation
[T]	Returns the activation time of the currently displayed alarm.
[f] [T]	Returns the current time.
[D]	Returns the activation date of the currently displayed alarm.
[R]	Returns the repeat interval of the currently displayed alarm.
[f] [R]	Resets the currently displayed alarm to its next future occurrence as determined by its repeat interval.
[M]	Displays the Alpha string (message or ↑↑/↑ label call), if any, of the currently displayed alarm.
[f] [C]	Clears (deletes) the currently displayed alarm from memory.
[←]	Exits the alarm catalog and Alarm Catalog keyboard and returns to a display of the X-register.
[ON]	Turns the computer off; cancels the alarm catalog.

If the computer is halted in the alarm catalog, and no key is pressed for about 2 minutes, the computer automatically exits the alarm catalog.

Although [ALMCAT] is programmable, the individual Alarm Catalog keyboard operations are not. If [ALMCAT] is executed from a program, that program resumes after the alarm catalog has been exited.

* General aspects of operation common to all catalogs are given in section 9. Pressing any other key besides [R/S] or [ON] during the running catalog will speed up the listing.

Clearing Alarms From Memory

Control alarms (including conditional alarms that become control alarms) clear themselves automatically after going off. Message alarms (including conditional alarms that become message alarms) are cleared by acknowledging them. In addition, there are general clearing operations for clearing one or more future and/or past-due alarm. Finally, agility is needed to cancel a repeating control alarm that has a very short repeat interval.

Clearing a Message Alarm as It Goes Off

If a message alarm is going off, acknowledging it will halt it and clear it from memory, unless it is a repeating alarm. In the case of a repeating message alarm in the process of going off, [shift] [C] will halt it and clear it from memory.

General Clearing Operations

There are various ways to delete any future and past-due alarms (that is, alarms that are not currently going off), as well as repeating control alarms.

- The easiest way to delete any one such alarm is with [shift] [C] on the Alarm Catalog keyboard (catalog 5; see the previous topic), but this method is *not programmable*.

The other alarm-clearing functions—[CLRALMS], [CLALMA], and [CLALMX]—are programmable. The latter two are especially useful if you want to have a program set a repeating alarm and then clear it later.

Clearing All Alarms ([CLRALMS]). [CLRALMS] (*clear alarms*) deletes all alarms from memory. It is sometimes used in conjunction with [RCLALM] (page 252) to delete all alarms, thereby recovering alarm memory space, *after* having individually recalled each alarm and stored its parameters.

Clearing an Alarm by Its Alpha String ([CLALMA]). [CLALMA] (*clear alarm by Alpha*) deletes the first alarm whose Alpha string matches the string in the Alpha register.

If there are no alarms with duplicate Alpha strings (that is, messages or ↑↑/↑ label calls), then [CLALMA] is the most foolproof way to clear an alarm. If more than one alarm has the same Alpha string, then only the first of them (as they are listed in the alarm catalog) will be cleared.

If the Alpha register is empty when [CLALMA] is executed, then [CLALMA] deletes the first alarm that has no message.

NO SUCH ALM results if there is no alarm with the given Alpha string.

Clearing an Alarm by Its Ordinal Number (CLALMX). CLALMX (*clear alarm by X*) deletes the alarm specified by the number in the X-register. The number refers to the ordinal position of the alarm in the alarm catalog.

Duplicate alarm Alpha strings pose no problem. However, you should keep in mind that an alarm's number can change (increase or decrease) anytime another alarm is set, reset, goes off, or is cleared.

CLALMX takes the absolute value of the integer part of the number in the X-register to be the alarm number. If, for example, the contents of X are 5.1 when CLALMX is executed, then the fifth alarm in the catalog would be cleared.

If $x = 0$ or $x > 999$, then **DATA ERROR** results when CLALMX is executed.

NO SUCH ALM results if there is no alarm of the given number.

Clearing Repeating Control Alarms

Usually, a repeating control alarm can be cleared using the above methods. However, if a control alarm has a repeat interval shorter than about 10 seconds (1 second is the minimum), it can be difficult to delete it from memory because of the time it takes to do so. That is, the alarm might be able to repeat and reset itself before you could cancel it, although CATALOG 5 can be executed quickly.

There are two other ways around this time limitation:

- Assign ALMCAT to a key on the User keyboard, so that it can be executed with one keystroke. The alarm catalog condition is all part of one function, so no alarm will interrupt it. You can then use ■ C to cancel the alarm in question.
- Use the "two-key rollover" technique to execute CATALOG 5 or another clearing function. By depressing the next key before releasing the previous key, the HP-41 keyboard is kept operating without a break. The alarm will not go off because it keeps waiting for the break between functions or keys. Using two-key rollover, there is always a function or key execution in progress. (The alarm keeps waiting for the end of a function, so it does not become past due.)

In a program, use an alarm-clearing function (CLALMA or CLALMX) to clear any repeating control alarm.

Past-Due Alarms

A past-due alarm is any alarm in memory having an alarm time that is earlier than the current time. The following information provides a basic description of past-due alarm operation. When more than one past-due alarm accumulates, there are rules governing the order of their automatic activation. These are described in appendix B, "More About Past-Due Alarms."

Creating Past-Due Alarms

A past-due alarm normally results if:

- A message alarm activates (goes off) without being acknowledged, or is acknowledged using [STO].
- A conditional alarm (↑) activates while the HP-41 is running a program.
- A conditional alarm (↑) activates as a message alarm (because the HP-41 is not off), which is then not acknowledged.

Note that a control alarm cannot normally become a past-due alarm since it always activates when its time comes due, then automatically clears or resets itself. However, a control alarm can become past due if its time is *bypassed*.

Bypassed Past-Due Alarms

A *bypassed* (also called *unactivated*) past-due alarm results if:

- Any future alarm is bypassed due to a change in the time.
- Any alarm is initially set to a past time.

This type of past-due alarm should be rare. It is the only type of past-due alarm for control alarms.

Automatic Reminder of Past-Due Alarms

Whenever you turn the HP-41 **on**, a pair of tones will sound if any past-due alarms exist. This is strictly a reminder; the alarms are not activated—so they have no effect—and they cannot be acknowledged.

The automatic reminder also sounds if you change the time or execute [XYZALM] when any *bypassed* past-due alarms exist or are created.

Activation of Past-Due Alarms

Whenever you try to turn the HP-41 **off** *or* use [ON] (*not* [CLOCK]), the HP-41 automatically activates past-due alarms.*

Automatic activation starts with the earliest alarm:

- Past-due message alarms go off.
- Past-due control alarms execute their designated programs.†

---

* After executing [CLOCK], subsequently pressing [ON] will not activate any remaining past-due alarms.

† The automatic activation of a control or conditional alarm momentarily turns off the HP-41 first. For this reason, no other past-due alarms—except bypassed ones—will activate subsequently, until you press [ON] or [ON] again. See appendix B.

- A past-due conditional alarm activates automatically as a control alarm (if there are no preceding past-due control alarms).* In this way, if the end of a program automatically turns off the HP-41 (using the [OFF] instruction), then any conditional alarm that came due while the program was running will automatically be executed when the program shuts off the computer.

Refer to appendix B for information about the automatic activation of multiple past-due alarms.

Automatic Clearing/Reset of Automatically Activated Past-Due Alarms. When a past-due message alarm is automatically activated, acknowledging it will clear it from memory—or reset it if it has a repeat interval. ([ON] will halt but *not* clear/reset an activating past-due alarm. See "Halting the Activation of Past-Due Alarms," below.)

A past-due control or conditional alarm that is automatically activated will also automatically clear (or reset, if it's a repeating alarm) itself. A repeating alarm is reset *to the future* using multiples of the repeat interval added to the original alarm time.

After acknowledgment of past-due *message* alarms, the HP-41 will complete the function that triggered the automatic activation; that is, it will shut off or display the clock.

Halting the Automatic Activation of Past-Due Alarms. If you press [ON] during the automatic activation cycle of a past-due *message* alarm, this aborts the entire series and cycle of past-due alarm activation. The alarm is *not* acknowledged. The unacknowledged and unactivated past-due alarms remain past due.† This provides a means of stopping what could be a long activation series so you can regain control of the computer but save the alarms. (This is also handy if you inadvertently set the time ahead and your future alarms become past due.)

If you press [ON] during the activation of a past-due *control or conditional* alarm, this will shut the HP-41 off, stopping whatever program that a control or conditional alarm had started. (This will trigger the activation of any remaining past-due alarms.) The interrupted control/conditional alarm will be cleared or reset, since it was activated.

Activating a Past-Due Conditional Alarm ([ALMNOW]). You can activate a single past-due conditional (or control) alarm—the *earliest* (oldest) one in memory—by executing [ALMNOW] (*alarm now*).

While a program is running, it is possible for a conditional (↑) alarm to come due and therefore become past-due. You can check for and activate one such alarm—or have the program do it—by executing [ALMNOW], provided there are no other past-due conditional or control alarms in memory. If executed from a program, [ALMNOW] operates as a subroutine.

---

* The automatic activation of a control or conditional alarm momentarily turns off the HP-41 first. For this reason, no other past-due alarms—except bypassed ones—will activate subsequently, until you press [ON] or [■] [ON] again. See appendix B.

† Using [ON] to halt an automatic activation cycle will also cancel the activation of any other alarm that happened to come due (for the first time) during the automatic, past-due activation cycle.

Application Programs for Setting Alarms

To use the following programs, first key them from the listings into program memory. There are bar-code versions of these programs in appendix J, "Bar Code for Programs." If you have an HP 82153A Wand, you can record these programs quickly from the bar code.

Using a Program to Set an Alarm (SETALM)

If you have trouble remembering exactly how to set a particular alarm, then the following program, SETALM, will help you. SETALM provides an easy way to set any kind of alarm, prompting you for the necessary input and placing that information into the correct register.

Input (when prompted): The alarm time, a message (for a message, control, or conditional alarm), the alarm date, and the alarm repeat interval, if any.

Result: A message, control, or conditional alarm will be set.

User Instructions:

1. Execute SETALM (SETALM).
2. In response to the prompt **TIME?**, enter the alarm time in ***HH.MMSS*** format. (If no response, the program ends.)
3. In response to the display **MESSAGE?**, enter a message (for a message alarm), or ↑ ↑ *global label* (for a control alarm), or ↑ *global label* (for a conditional alarm). If you want a message alarm to display only the time and date, just press R/S .
4. In response to the display **DATE?**, enter the date in ***MM.DDYYYY*** or ***DD.MMYYYY*** format. If you want the current date, just press R/S .
5. In response to the display **RESET?**, enter the repeat interval in ***HHHH.MMSS*** format. If you do not want a repeating alarm, just press R/S .

This program will alter any information you had in the stack and the Alpha register before running this program, and it clears flag 22.

Program Listing

```
01♦LBL "SETALM"
02 CF 22
03 "TIME?"
04 PROMPT
05 FC?C 22
06 RTN
```

Clears flag 22, the numeric data input flag.

Stops and asks for the alarm time. If you do not enter a time (flag 22 tests clear), then the program ends. (After entering the time, restart the program with R/S .)

```
07 "DATE?"
08 ASTO T
09 "RESET?"
10 ASTO Y
```

Stores the message **DATE?** in the T-register and the message **RESET?** in the Y-register.*

```
11 "MESSAGE?"
12 AVIEW
13 CLA
14 AON
15 STOP
```

Displays the message **MESSAGE?** from the Alpha register and stops for Alpha input. (Clears the Alpha register—**MESSAGE?** still displayed—activates the Alpha keyboard, and stops.) If you enter no message, the alarm will display the time and date since line 13 clears the Alpha register.

```
16 AOFF
```

```
17 VIEW T
18 STOP
```

Displays **DATE?** and stops for input.*

```
19 FC?C 22
20 0
```

If you do not enter a date (no numeric input; flag 22 tests clear), 0 is used (current date).

```
21 VIEW Z
22 STOP
```

Displays **RESET?** (lifted from Y-register) and stops for repeat interval.*

```
23 FC?C 22
24 0
```

If no repeat interval entered, 0 is used.

```
25 X<> Z
26 XYZALM
27 END
```

Brings the time (which has been lifted into the Z-register) back to the X-register and sets the alarm.*

Setting an Alarm Relative to the Current Time (ALMREL)

The following program example illustrates several programming techniques—including extensive stack manipulation—to create an all-purpose program (ALMREL) to set an alarm relative to the current time. This is useful when you want to set an alarm for a certain period of time from the present, rather than for a particular clock time.

Input (when prompted):

1. The time "offset" (the number of hours, minutes, seconds from the present) as ***HHHH.MMSS***. Note this can represent more than one day.
2. Alpha string for alarm (message or ↑↑/↑ label call).

Result: A message, control, or conditional alarm.

* The functions [ASTO], [AVIEW], and [X<>] are parameter functions, and display two input cues when executed: for example, X<> __. Press [·] [Z] to specify the Z-register as a parameter.

User Instructions:

1. Execute ALMREL (ALMREL).
2. In response to the display **+ HH.MMSS?**, enter the time offset ***HHHH.MMSS***, then press R/S .
3. In response to the display **MESSAGE?**, enter alarm Alpha string (or nothing), then press R/S .

Program ALMREL begins to calculate the alarm time when you press R/S after **MESSAGE?**. The time offset can be as short as 0000.0003 (3 seconds) or as long as 9999.595999. The program checks for and rejects non-numeric or negative input for ***HH.MMSS***. (Flag 22 is set when numeric data is entered.)

This program will alter any information you had in the stack and the Alpha register before running this program, and it clears flag 22.

Program Listing

```
01♦LBL "ALMREL"
02 CF 22
03 "+ HH.MMSS?"
04 PROMPT
05 FC?C 22
06 RTN
07 X<0?
08 RTN
09 "MESSAGE?"
10 AVIEW
11 CLA
12 AON
13 STOP
14 AOFF
15 TIME
16 HMS+
17 ENTER↑
18 ENTER↑
19 24
20 /
21 INT
22 DATE
23 X<>Y
24 DATE+
```

Clears flag 22 so it can be tested later.

Stops and prompts for time offset. (Restart the program with R/S).

Tests for non-numeric input.
If non-numeric (flag 22 is clear), start over!

Tests for negative input.
If negative, start over!

Stops after asking for input for the alarm's Alpha string. The program clears the Alpha register and activates the Alpha keyboard (AON), then deactivates it (AOFF) when you restart the program. If you want a message, just type it in and press R/S . If not, just press R/S .

Recalls current time.
Adds current time to offset time.

Puts sum (the new time, in hours) in X-, Y-, and Z-registers.

Finds the number of whole days away the new time is (the whole days of time offset).

Recalls the current date.
Days of offset in X; current date in Y.
Finds the new date for the alarm: the sum of the current date and the number of offset days.

```
25 LASTX
26 24
27 *
28 ST- Z

29 CLX
30 STO T
31 RDN

32 X<>Y
33 XYZALM
34 END
```

Recalls the number of offset days and converts that figure to hours. That figure is then subtracted from the total number of offset hours in Z, yielding the number of hours of offset *beyond whole days*. This value represents the time for the alarm.

Puts zero in the X-register and copies zero into the T-register. After the roll-down, this will represent the repeat interval (none) in Z. The new time (from Z) moves into Y; the new date (from Y) moves into X.

Date into Y; time into X. The stack is now set up to set the alarm.

Section 17

Stopwatch Operation

Contents

The *stopwatch* function, SW (or SWPT), encompasses an entire mode of operation. It turns the HP-41 into a stopwatch and redefines the keyboard for stopwatch operations. The internal timer used for the stopwatch is separate from the clock. The stopwatch can run even when it's not displayed, and even when the HP-41 is off. It can continue to run unimpeded while you execute other functions or programs.

An introduction to stopwatch operation, with examples, is presented in section 6. These functions are also summarized in the Function Table "Time Functions."

The Stopwatch Keyboard (SW)

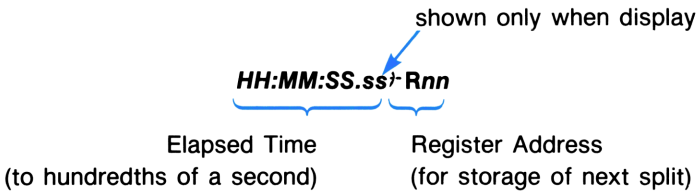
The HP-41CX comes with a keyboard overlay, which marks all the active functions on the Stopwatch keyboard. A diagram of the keyboard is also included below and in the Quick Reference Guide. Any key not identified on the overlay and diagram is inactive as long as the Stopwatch keyboard is in effect.

- Executing [SW] or [SWPT] activates the stopwatch display and redefines the keyboard. [SW] resets the register pointers to zero and sets the display to show regular splits; [SWPT] sets the pointers as you specify them (page 274).
- Pressing [EXIT] deactivates the Stopwatch keyboard, *but does not automatically stop a running stopwatch.* The display returns to the X-register. This operation is not programmable.
- [ON] will turn the HP-41 off (and deactivate the Stopwatch keyboard), even if the stopwatch is still running. *It will not stop the stopwatch.*

Although [SW] and [SWPT] are programmable, none of the operations on the Stopwatch *keyboard* are programmable. There are other programmable stopwatch functions discussed later in this section.

The Stopwatch Display

While the Stopwatch keyboard is active, the display has the form:



Note: The stopwatch display consumes as much power as a running program. Refer to "Power Consumption" in appendix G. *The HP-41CX will not automatically turn off (time out) as long as the Stopwatch keyboard is active.*

The elapsed time shown in the display does *not* clear whenever you activate and deactivate the Stopwatch keyboard. (Just as deactivating the Stopwatch keyboard does not stop a running time, either.) However, activating the Stopwatch keyboard with [SW] does reset the pointer display to ↑R00.

Pressing [CLEAR] resets the stopped stopwatch to 00:00:00.00. When the stopwatch time passes 99:59:59.99, it automatically starts again from zero. There is no stopwatch function to clear stored times (splits). A time stored in a register will *replace* any previously stored value there. To put zero in a register, store a zero time in it, or, outside of the stopwatch, use [CLRGX] to clear more than one register.

Pressing any undefined Stopwatch key will freeze the display while the key is depressed without halting the stopwatch itself. This will show you the elapsed time to tenths of a second. Only when the stopwatch is halted does it show the full eight digits.

Stopwatch Keyboard Operations

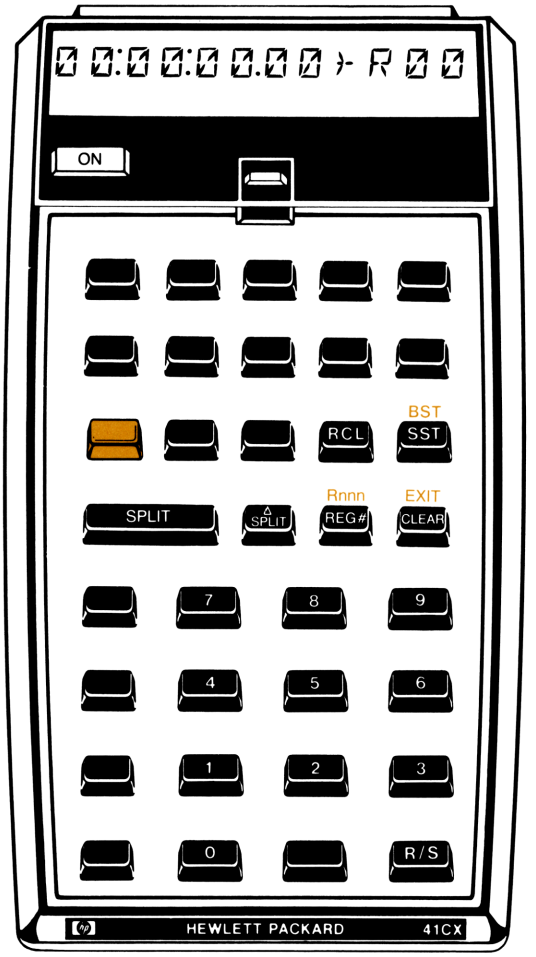
The following operations comprise the Stopwatch keyboard, which is activated by the execution of [SW] or [SWPT]. In addition, executing [SW] resets both storage (STO) and recall (RCL) pointers to zero, and makes Regular Split Storage mode the default mode of operation (STO R00).

The Stopwatch Keyboard

Key	Operation
[R/S]	*Run/stop.* Starts and stops the stopwatch; does not reset the stopwatch or the register pointers. (Also cancels Recall mode.)
[CLEAR]	Resets a halted stopwatch to 00:00:00.00. Does not affect the register pointer. (If Recall mode is set, [CLEAR] cancels it and does not reset the stopwatch.)
[EXIT]	Deactivates the Stopwatch keyboard. *This will not stop a running stopwatch.*
[SPLIT]	Stores the current stopwatch time in the indicated data storage register (STO **R*nn***), and increments the pointer. (Also cancels Recall mode.)
[RCL]	*Recall.* Toggles between Recall mode and Storage mode. Recall mode displays the split stored in the indicated register (RCL **R*nn***). (Recall mode is also cancelled by [R/S], [SPLIT], and [CLEAR].)
[△SPLIT]	*Delta split.* Toggles between Delta Split mode (**D**) and Regular Split mode (**R**). Delta Split mode displays the difference between the most recent split and the split in the preceding register. This display changes with each execution of [SPLIT]. Can also be used in Recall mode. Delta Split mode is also cancelled when Stopwatch keyboard is deactivated.
Digit Keys*	Used only to specify storage register addresses (*nn*, or *nnn* with [Rnnn]). Keying in a number moves the register pointer to that address. Be sure to specify both or all three digits, as required.
[SST]*	*Single step.* Moves the register pointer to the next register without taking a split.
[BST]*	*Back step.* Moves the register pointer to the previous register without taking a split.
[REG#]	*Register number.* Suppresses/restores the display of the register pointer. Toggle switch.
[Rnnn]	Changes the register-pointer display from two digits to three digits (and back) in order to display $R_{(100)}$ to $R_{(319)}$. Toggle switch.
Any Other Key	Pressing any other key freezes the last stopwatch time in the display for as long as the key is depressed. The stopwatch itself continues to run.

* These functions will change the register pointer whether the stopwatch is running or halted.

The Active Keys on the Stopwatch Keyboard



The Register Pointers

The register pointers, represented in the display as **Rnn** and **Dnn**, give the address, *nn*, of the current data storage register.

The **R** means *regular* split; the **D** means *delta* split (the difference between two splits). The ⊦ means splits will be stored; the ⁼ means splits (or delta splits) will be displayed.

Split Storage	
⊦**R***nn*	Store split.
⊦**D***nn*	Store split; display difference.
Split Recall	
⁼**R***nn*	Recall split.
⁼**D***nn*	Recall split difference.

The Current Register. *The register pointers always reset to 00 following the execution of* [SW] (even though the time does not reset to zero). The current register, *nn*, has the following functions:

- With ⊦**R***nn* (storing regular splits), *nn* represents the register into which the *next* split taken ([SPLIT]) will be stored.
- With ⊦**D***nn* (storing delta splits), *nn* represents the storage register for the next split *and* it represents the second of two successive registers whose difference will be found and displayed (using [SPLIT]).
- With ⁼**R***nn* (recalling regular splits), *nn* represents the register whose contents are currently being displayed. (The display is static, though the stopwatch itself might be running.)
- With ⁼**D***nn* (recalling delta splits), *nn* represents the second of two successive registers whose difference is being displayed. (The display is static, though the stopwatch might be running.)

Changing the Register Pointers. The ⊦ (storage) register pointer advances automatically every time you take a split. You can also change it manually, using [SST], [BST], and the digit keys as shown in "The Stopwatch Keyboard". The ⁼ (recall) register pointer is changed manually only.

> **Note:** The register pointers for split Storage (⊦) and Recall modes (⁼) are maintained separately, while the Regular Split (**R**) and the Delta Split modes (**D**) are different display modes for the same register pointer. Both the ⊦ and the ⁼ register pointers are reset to 00 by [SW].

Three-Digit Display of the Register Pointer ([Rnnn]). The register-pointer display will automatically switch from two digits to three digits when it advances from R_{99} to $R_{(100)}$. To switch the display manually between the two- and three-digit display, use [Rnnn]. (If you switch back to a two-digit pointer display when the current register has three digits, the leftmost register digit will be dropped.) A three-digit pointer display suppresses the rightmost digit in the display of the stopwatch time, though it is retained internally.

Suppression of the Register-Pointer Display. Pressing [REG#] suppresses (and restores) the display of the register pointer, though the register pointer is maintained internally. This allows you to view the obscured rightmost time digit in the cases when the register pointer has three digits or the stopwatch time is negative.

General Stopwatch Operation With Splits

Splits (stopwatch timings) taken on the HP-41 are automatically stored in sequential *data storage registers*, the same registers you use for regular data storage. Sequential splits are stored as *accumulated times* unless you reset the stopwatch to zero ([CLEAR]). You can store as many splits as you have storage registers available. Any other data already in the current register is replaced when a split is taken, just as any stored splits can be accessed and replaced when you use those same registers without the Stopwatch keyboard.

Splits are displayed as ***HH:MM:SS.s*** or ***HH:MM:SS.ss***, but they are stored as ***HH.MMSSss***. Therefore, if you have stored a split in a register, then exit the Stopwatch keyboard, the contents of that register will be ***HH.MMSSss***. Conversely, if a register contains a value that is not a split, and those contents are recalled with the Stopwatch keyboard, the HP-41 will attempt to display that value in the form ***HH:MM:SS.ss***. If it can't—because ***HH*** > 99—then **ERROR** ⁻ **R*nn*** results. (Values of ***MM*** and ***SS*** greater than 59 do not cause errors.) To clear **ERROR** ⁻ **R*nn***, move the pointer to a register with a valid split. Pressing [CLEAR] will cancel Recall mode.

There is a diagram of the modes of stopwatch operation on page 76 in section 6.

The Display During Split-Taking

While the [SPLIT] key is held down during Regular Split mode (**R*nn***), the stopwatch display shows the split that was just taken and the register it was stored into. (The stopwatch itself does not stop). When you release [SPLIT], the running display resumes and the register pointer advances.

If Delta Split mode is active (**D*nn***), [SPLIT] operation is the same as usual. However, the display while [SPLIT] is being held down is different: it shows the difference between the split just taken and the split in the previous storage register. The register shown is the location of the split just taken. (Refer also to "Viewing Delta Splits," below.)

Register-Pointer Limit

If the current register-pointer (**R*nn*** or **D*nn***) is moved—either automatically or manually (but not with [BST])—to the last available data storage register or beyond, the HP-41 beeps. (This is true whether the stopwatch is running or not.)

Furthermore, if the stopwatch is in Recall mode (⁻) when the pointer is moved to a nonexistent register, this will cancel the Stopwatch keyboard and display **NONEXISTENT**. This also occurs during Storage mode (⁻) if the execution of [SPLIT] attempts to store a split in a nonexistent storage register.

Recalling Splits (RCL)

Pressing RCL toggles the stopwatch and its display between Recall mode (⁚) and Storage mode (⊦). In Recall mode, the display shows the contents stored in the indicated register. A running stopwatch will continue to run during Recall mode. To view the splits stored in other registers, just change the register pointer (using SST, BST, or the digit keys).

When you press RCL again, or press SPLIT, R/S, or CLEAR, the display returns to the regular stopwatch. Remember that the register pointers for split storage and split recall are maintained separately, so as the display transfers between Storage (⊦) and Recall (⁚) modes, the respective pointer address (*nn* or *nnn*) will take up where you last left it. (Refer to "The Register Pointers.") This allows you, for instance, to take several splits, switch to Recall mode (with the stopwatch running or stopped) to review the splits you just stored, then switch back to regular Storage mode to resume storing splits in the register that follows the last one you used for split storage.

Viewing Delta Splits (ΔSPLIT)

Pressing ΔSPLIT activates Delta Split mode: the stopwatch display shows delta splits (**D**) rather than regular splits (**R**).

- A *delta split* is the difference between the current split (the one in **Dnn**) and the split in the immediately preceding register.
- If the register pointer is **D00**, then the delta-split display just shows the current split.

Delta Split mode allows you to display the difference between two splits without interrupting a running stopwatch. It can be used during split Storage (⊦**Dnn**) or split Recall (⁚**Dnn**) modes.

Using Delta Split mode with either SPLIT or RCL allows you to compare quickly the difference between two related time measurements. Using delta splits also allows you to take timings that do not overlap at all but are too close together to have time to reset the stopwatch. For instance, you could store both the start- and the stop-times of a series of closely spaced, non-overlapping events, so that you would end up with a series of paired start- and stop-times. By recalling the split difference, you'd see the actual time of the event.

Storage Versus Recall. Delta splits are neither stored nor recalled, they are calculated. Pressing SPLIT takes regular splits even when Delta Split mode is active (⊦**Dnn**), but the frozen display (while the SPLIT key is down) *shows* delta splits. "Recalling" splits in Delta Split mode (⁚**Dnn**) does *not* really recall delta splits, it displays delta splits calculated from stored splits.

Therefore, Delta Split mode does not alter the operation of either SPLIT or RCL (Recall mode); it just means that any static display you see represents a delta split and not a regular split. (The running stopwatch display is not affected.)

Negative Delta Splits and Errors. If the second of two adjacent splits is *not* larger than the first, then the display depends on whether splits are being stored or recalled: if splits are being stored (→*Dnn*), then the delta-split display shows just the contents of the current register; if delta splits are being recalled (←*Dnn*), then **ERROR** ←*Dnn* results.

> **Note:** To clear **ERROR** ←*Dnn* or **ERROR** ←*Rnn*, change the register pointer address. (Pressing CLEAR will cancel Recall mode without clearing the error condition.)

If, during Recall mode in Delta Split mode (←*Dnn*), one of the two registers involved contains a value that does not fit the form ***HH.MMSSss*** (see page 271), then **ERROR** ←*Dnn* results.

Programmable Stopwatch Functions

There are six programmable stopwatch functions: the two that activate the Stopwatch keyboard (SW and SWPT) and four others (RUNSW, STOPSW, SETSW, and RCLSW) to manipulate the stopwatch when the stopwatch keyboard is not active. You can use these latter four functions to set up the stopwatch or run an internal timer during the execution of a program. (You cannot execute these functions from the Stopwatch keyboard.)

Any stopwatch times that you put into or recall into the X-register should be of the form ±***HH.MMSShh***.

Starting and Stopping the Stopwatch (RUNSW, STOPSW)

Run Stopwatch. RUNSW starts the stopwatch running.

Stop Stopwatch. STOPSW stops the stopwatch.

Setting and Recalling the Current Stopwatch Time (SETSW, RCLSW)

Set Stopwatch. SETSW sets the stopwatch to the time (±***HH.MMSSss***) specified in the X-register.

Any specified time outside the range −99.595999 to +99.595999 is invalid and will cause a **DATA ERROR**. Any digits beyond the **ss** places are just ignored.

SETSW will not stop a running stopwatch, but it will reset its time as specified.

Recall Stopwatch. RCLSW recalls the current stopwatch time to the X-register in the form ***HH.MMSSss***, lifting the stack (unless stack lift is disabled).

Setting Up the Stopwatch and the Stopwatch Pointers (SWPT)

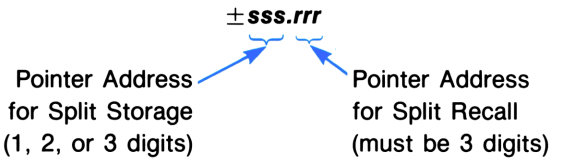
The function SWPT (*stopwatch and pointers*) acts like SW, but in addition sets the stopwatch pointers. This allows a program to set the pointers to avoid taking splits that will write over other data stored by the program.

Using SWPT:

- Upon activating the Stopwatch keyboard, the storage register-pointer (→Rnn, →Dnn) and the recall register-pointer (=Rnn, =Dnn) are set as specified by the *sss.rrr* value in the X-register.
- Upon deactivating the Stopwatch keyboard, the current pointer values are returned to the X-register in the form *sss.rrr*. The input *sss.rrr* value is saved in the LAST X register.

The output value of *sss* can be used to figure how many splits were taken, *assuming splits were taken in sequential registers* and the split-storage register pointer was not manually changed.

Combined Pointer Value in X-Register



A *positive or zero value* specified for *sss.rrr* sets Regular Split mode (→Rnn); a *negative value* for *sss.rrr* sets the stopwatch to Delta Split mode (→Dnn).

(To set the stopwatch to Delta Split mode and also set both pointers to zero, specify a number between −0.001 and zero. If, when the Stopwatch keyboard is deactivated, both pointers are set to zero and Delta Split mode is in effect, the value −0.0000001 will be returned to the X-register.)

In the unlikely case that upon deactivating the Stopwatch keyboard one pointer is undefined, the pointer value (*sss* or *rrr*) returned to the X-register will be zero. (This can happen if the Stopwatch keyboard is deactivated while the register pointer address is incomplete.)

The Stopwatch as a Countdown Timer

If the stopwatch is set to a *negative time* (use SETSW) and then runs, it will set off a timer alarm when it reaches 00:00:00.00.

- If the Stopwatch keyboard and display are not active, the timer alarm will sound like a message alarm and display **TIMER ALARM**. You can stop the alarm by pressing any key, but acknowledgment is not necessary as this alarm is not stored in memory and cannot become past due.
- If the Stopwatch keyboard *is* active, then the timer alarm will merely sound two tones (the stopwatch display is not interrupted).

In neither case will the running stopwatch automatically stop; the stopwatch starts counting up after passing through zero. To stop the stopwatch, use [STOPSW] (not from the Stopwatch keyboard) or [R/S] (from the Stopwatch keyboard).

Printing Stored Splits

If you have an HP 82143A Printer or an HP 82162A HP-IL Printer and want to print out a stored split, you can do so with the [ATIME24] function (section 15) in conjunction with the [PRA] function (*print Alpha*; see your printer or HP-IL manual). You cannot print out splits while the Stopwatch keyboard is active.

Since split differences (delta splits) are not stored, they must be recalculated if you want to print them out. You can do this using the [HMS−] function (not from the Stopwatch keyboard):

1. Recall the later split value from its storage register.
2. Recall the earlier split value from its storage register.
3. Execute [HMS−].
4. Clear the Alpha register, if desired.
5. Execute [ATIME24].
6. Execute [PRA].

To print out regular splits, just skip steps 1 through 3 and recall the desired split value.

Example—A Stopwatch Program (SPLITS)

The following program sets up the stopwatch in preparation for taking timed splits, activates the Stopwatch keyboard, and—when the Stopwatch keyboard is deactivated—prints out a specified group of stored splits. (A printer must be attached for this program.) The splits are printed out in the format ***HH:MM:SS.ss***. The value recalled from each register must be less than 100, otherwise an error results.

To use the following program, first key it from the listing into program memory. There is a bar-code version of this program in appendix J, "Bar Code for Programs." If you have an HP 82153A Wand, you can record this program quickly from the bar code.

Input (when prompted):

1. The number of the first register from which you want a split to be printed.
2. The number of the last register from which you want a split to be printed.

Result: A list of each storage register, from the first to the last, and the value stored in it. The value will be printed in time format.

User Instructions:

1. Execute SPLITS ([SPLITS]).
2. In response to the stopwatch display, start taking as many splits as desired. You can start from any register you want, but the splits must be taken in sequential registers.
3. When done taking splits, press [EXIT].
4. In response to the message **FIRST REG?**, enter the first register whose contents (split) you want printed. If you want to start from R_{00}, you don't have to enter a number. Press [R/S].
5. In response to the message **LAST REG?**, enter the last register whose contents you want printed. Then press [R/S].

Program Listing

01♦LBL "SPLITS"	
02 STOPSW	Makes sure stopwatch is stopped.
03 0 04 SETSW	Resets stopwatch to 00:00:00.00.
05 SW	Turns on Stopwatch keyboard. After the desired number of splits is taken and the Stopwatch keyboard is cancelled, the program continues.
06 "FIRST REG?" 07 PROMPT 08 "LAST REG?" 09 PROMPT	If no other number is entered, the first register will be R_{00} (owing to the zero in line 03).
10 ADV	Moves printer paper.
11 RCLFLAG 12 X<> Z	Saves the current flag settings in the Z-register.
13♦LBL 00	
14 FIX 0	No fractional part for a register number.
15 CF 29	In combination with [FIX] 0, this suppresses the radix mark.
16 "R"	Starts to form output string (**R** for register).
17 100 18 X<=Y? 19 GTO 01 20 "⊢ " 21 SQRT 22 X>Y? 23 "⊢0"	Lines **17 through 23** are for the output format: if the register number is less than 100, then insert a space between the **R** and its number. If the register number is less than 10, then add a zero in front of the register number. This will result in a printed output with all register numbers and their contents aligned.

```
24♦LBL 01
25 RDN
26 ARCL X
27 "⊢ ="
28 RCL IND X
29 FIX 6
30 ATIME24
31 PRA
32 RDN
33 1
34 +
35 X<=Y?
36 GTO 00
37 RCL Z
38 STOFLAG
39 CLX
40 END
```

Brings number of first register into X.

Takes the register number in X and appends a copy of that number to the string in the Alpha register.

Recalls the value from the register whose address is in X.

Appends that value in time format to the Alpha register.

Prints the entire string in the Alpha register.

Increments the number of the first register by one and continues the program if the number of the first register is still less than or equal to the number of the last register.

Restores the original flag settings, which include the display formats.

Clears the X-register.

Sample Output: Following is an example of a printer output for splits taken in registers R_{00} through R_{10}.

```
            XEQ "SPLITS"
FIRST REG?
                        RUN
LAST REG?
          10.0000       RUN

R 00 =00:00:07.62
R 01 =00:00:09.35
R 02 =00:00:10.20
R 03 =00:00:12.24
R 04 =00:00:13.99
R 05 =00:00:07.06
R 06 =00:00:08.90
R 07 =00:00:10.54
R 08 =00:00:12.76
R 09 =00:00:13.77
R 10 =00:00:17.89
```

Part V:
Programming in Detail

Section 18

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, key redefinition, or alarm.
- 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, so that catalog 1 will display the number of bytes in the program.)
- 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:

- Check how many registers are allocated to data storage using SIZE?, and then allocate fewer registers using SIZE or PSIZE.
- Delete complete programs, using CLP or PCLPS. (You can first use SAVEP to save copies of these programs in extended memory.)
- Clear one or more alarms.
- 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.
- Setting a control alarm to execute the program at a specified time.

The **PRGM** annunciator appears in the display while a program is running. Unless a function like [AVIEW] displays a message, or a function like [ED] activates a special keyboard and display, 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 9, "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] (but [PCLPS] is programmable).

- Positioning operations:

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

- All catalogs (but [EMDIR] and [ALMCAT] are programmable).

- 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] (but [PASN] is programmable).

  [SIZE] (but [PSIZE] is programmable).

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.

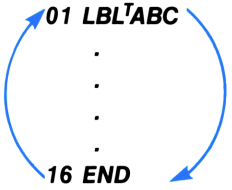
If a program doesn't contain any global labels, follow the five steps above to position the computer to the program's END statement. (When two END statements appear sequentially, the second END statement belongs to a program without global labels.) You should then insert a global label at the start of 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 20, "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.

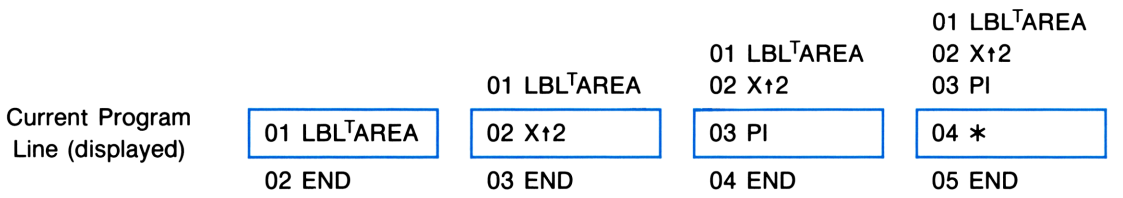
In the previous example, lines 02, 03, and 04 are deleted one by one. Alternatively you could position the computer to line 02 and execute [DEL] 003. This deletes lines 02, 03, and 04, leaving the computer positioned to the previous line (line 01). The line number of each subsequent instruction is reduced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



Clearing Programs

There are two functions that clear programs. [CLP] *(clear program)* clears one program and is not programmable; [PCLPS] *(programmable clear programs)* can clear several programs and is programmable.

Using CLP

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.

Using PCLPS

To clear a program and all subsequent programs in program memory:

1. Place any global label from the program into the Alpha register.
2. Execute PCLPS.

Executing PCLPS when the Alpha register is empty clears the current program and all subsequent programs.

Section 19

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 20, "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.

The first eight flags (00 through 07) can be interpreted as the eight bits in a byte, and that byte can be transformed into a number in the X-register. This process is discussed after the control and system flags.

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 VIEW and AVIEW are executed, depending on whether an output device is present. For details, refer to appendix D, "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.

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 9.

Program Examples. The programs TR and Σ in section 22 use flags 22 and 23 (Data Input) when prompting the user with default values for time, date, or job name. The user can either key in an alternative value and then press [R/S], or else confirm the displayed value by simply pressing [R/S]. To determine whether the user keyed in an alternative value, the programs clear the appropriate Data Input flag before prompting and then test it afterwards. If the user keyed in an alternative value before pressing [R/S], the Data Input flag will be set.

The programs use flag 25 (Error Ignore) when accessing records in the text file TRECS. To access all records without knowing the number of records in advance, the programs contain loops that act on one record, set flag 25 before accessing the next record, and then test flag 25 afterwards. If there isn't another record, an error occurs and flag 25 is cleared, so the program can tell whether to exit the loop by testing flag 25.

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. You can save and restore certain options that are encoded by the computer as flags. This allows a program that sets options to restore the previous conditions when it is completed. The functions to do so, [RCLFLAG] and [STOFLAG], are described at the end of this section.

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:

- Flag 31 represents the date format, described on page 242.
- Flags 36 through 39 represent the number of displayed digits, described on pages 160 and 161.
- Flags 40 and 41 represent the display format, described on pages 160 and 161.
- Flags 42 and 43 represent the angular mode, described on page 186.

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 D, "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.

Flags and the X-Register

There are three reasons to move data between the flags and the X-register:

1. To save and restore options such as display format, which are encoded by the computer as control or system flags.
2. To keep multiple copies of a group of user flags, with only one copy active as flags at one time.
3. To transform information represented by user flags into a number, and vice versa.

There are two ways to move data between the flags and the X-register:

- [X<>F] *(X exchange flags)* exchanges the status of flags 00 through 07 with a number from 0 through 255 in the X-register. Thus, the status of flags 00 through 07 can be saved as a number and later restored (by [X<>F] again), or the number can be used in other ways (like calculations or branching). However, [X<>F] cannot affect control or system flags.
- [RCLFLAG] *(recall flags)* and [STOFLAG] *(restore flags)* can save and restore flags 00 through 43, so they can handle user, control, or system flags. However, they can't transform flag status into a number.

In summary, only [RCLFLAG] and [STOFLAG] can save and restore control and system flags, and only [X<>F] can transform the status of user flags into a usable number, but either can be used for multiple copies of user flags. First described is [X<>F] and its use for multiple copies of user flags, followed by [RCLFLAG] and [STOFLAG] and a comparison of their use for multiple copies of user flags.

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
31	Date Format	Clear	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	?	?

Flags and Numbers as Bytes

A byte, as a quantity of information, is the key to the correspondence between flags and numbers. A byte comprises eight bits, or binary digits: 00010100, 10001100, 00100000, and 00001111 are examples of bytes.

- A byte can be interpreted as eight flags, each 0 or 1 being the status of a particular flag.
- A byte can also be interpreted as a number, the sum of powers of 2.

By interpreting flags 00 through 07 as a byte and then interpreting that byte as a number, you can translate the status of eight flags into a unique number from 0 through 255. Conversely, you can specify the status of all eight flags at once by specifying a number from 0 through 255 and then translating it into the status of flags 00 through 07.

Decimal Values and Flags 00 through 07

Flag Number	07	06	05	04	03	02	01	00
Flag Status	Set or Clear	Set or Clear	Set or Clear	Set or Clear	Set or Clear	Set or Clear	Set or Clear	Set or Clear
Binary Value	2^7 or Zero	2^6 or Zero	2^5 or Zero	2^4 or Zero	2^3 or Zero	2^2 or Zero	2^1 or Zero	2^0 or Zero
Decimal Value	128 or Zero	64 or Zero	32 or Zero	16 or Zero	8 or Zero	4 or Zero	2 or Zero	1 or Zero

Example. Below are four particular bytes, each shown with its corresponding flag status and decimal value. Any flag from 00 through 07 that is not indicated as set is clear.

Flags That Are Set	Byte	Decimal Value
04, 02	00010100	16 + 4 = 20
07, 03, 02	10001100	128 + 8 + 4 = 140
05	00100000	32
03, 02, 01, 00	00001111	8 + 4 + 2 + 1 = 15

Using X<>F

When you execute X<>F:

- The status of flags 00 through 07 is transformed into a number from 0 through 255 and placed in the X-register.
- The number in the X-register, which must be from 0 through 255, is transformed into the status of flags 00 through 07. The sign and fractional part of the number in the X-register are ignored.

Multiple Copies of Flags

You can effectively increase the number of user flags by saving and restoring the status of the actual flags. When you save the current status of the actual flags, you create a "frozen" copy of those flags. You can later restore the actual flags to the status that was frozen in the copy, even though you might have used the actual flags for something entirely different in the meantime. In particular, you could have cleared the flags, encoded a new set of data in them, and saved the resulting flag status. This would give you *two* copies of the flag status, representing two distinct sets of data.

The following steps demonstrate this process. The first set of data is encoded by setting flags 00 and 02 ("first flag status") and is saved in R_{00}; the second set of data is encoded by setting flags 01 and 03 ("second flag status") and is saved in R_{01}.

01 0	Clear flags 00 through 07.
02 X<>F	
03 SF 00	Set the first flag status.
04 SF 02	
05 0	Move the first flag status to the X-register and clear flags 00 through 07.
06 X<>F	
07 STO 00	Save the first flag status in R_{00}.
08 SF 01	Set the second flag status.
09 SF 03	
10 RCL 00	Recall the first flag status to the X-register.
11 X<>F	Move the second flag status to the X-register and restore the first flag status.
12 STO 01	Save the second flag status in R_{01}.

This doubles the effective number of flags by encoding twice as much information, and you can create more copies as needed for additional sets of data. However, you can access only one set of data at a time. When you need to access a different set of data, you must store the current flag status and recall the desired flag status before you can test or manipulate the actual flags.

Example. Suppose that a program calls five subroutines and that each subroutine employs several flags. Before calling any subroutines, the program can create a copy of the initial flag status for each subroutine and store each copy in a separate register. Then whenever a subroutine is called, it can start by restoring its own flag copy, test and manipulate the flags, and finally save the current flag status as its updated flag copy. In this way, the program can call subroutines in any order and any number of times, and the subroutines won't interfere with each other's operation.

Example. Suppose you're analyzing the incomes of individuals by sex, age, and education. This can be encoded by flags as follows:

Sex: Flag 00 set = female, clear = male.
Age: Flag 01 set = age $<$ 20.
Flag 02 set = 20 $\leqslant$ age $<$ 35.
Flag 03 set = 35 $\leqslant$ age $<$ 50.
Flag 04 set = 50 $\leqslant$ age.
Education: Flag 05 set = High school.
Flag 06 set = College.
Flag 07 set = Graduate Studies.

Each individual can now be characterized by two numbers: by a number from 0 through 255 representing the status of flags 00 through 07; and by income.

To accumulate statistical data on a subset of the entire group of individuals, a program can test the flag copy for each individual in succession, accumulating only the incomes of those in the subset. For example, if the subset is defined as all men—regardless of age and education—you would restore the actual flags to correspond to the flag copy for each individual, test flag 00, and accumulate that individual's income if flag 00 is clear.

However, suppose that the subset is defined as women between 35 and 50 with a graduate degree. Because this definition involves all eight flags, you don't need to test separate flags; you can simply test the number that represents the status of all 8 flags. "Woman between 35 and 50 with a graduate degree" corresponds to flags 00, 03, and 07 set, which corresponds to a decimal value of 137. You would test the decimal value for each individual's flag copy and accumulate that individual's income if the decimal value equals 137.

Using RCLFLAG and STOFLAG

You can save the status of flags 00 through 43 using RCLFLAG *(recall flags)* and then restore some or all of them using STOFLAG *(restore flags)*. These two functions are useful only as a pair; the result of RCLFLAG can be used only by STOFLAG, and STOFLAG works only on data obtained by RCLFLAG.

Using RCLFLAG. When you execute RCLFLAG, the status of flags 00 through 43 is recalled to the X-register. This flag data can then be stored in a register in main or extended memory. (The display of the status data is meaningless.) Like RCL, RCLFLAG raises the stack unless stack lift is disabled.

Using [STOFLAG]**.** You can use [STOFLAG] to restore all or some of flags from 00 through 43:

- To restore flags 00 through 43, recall the flag-status data to the X-register and execute [STOFLAG].
- To restore a block of flags *bb (begin)* through *ee (end),* recall the flag status data as above and then place the control number *bb.ee* in the X-register. (This lifts the flag status data to the Y-register.) Then execute [STOFLAG].

Like [STO], [STOFLAG] doesn't change the stack. If $bb \geqslant ee$, only flag *bb* is restored.

Example. Suppose that a program rounds numbers to the nearest integer (requiring a [RND] instruction and a display format of [FIX] 0), but you want the user's choice of display formats to be in effect otherwise.

1. After input is completed with the user's choice of display format, use [RCLFLAG] to recall the status of flags 00 through 43 to the X-register, and then use [STO] *nn* to store the flag-status data in R_{nn}.
2. Use [FIX] 0 to specify "no decimal places."
3. Use [RND] to round the number in the X-register.
4. Before starting output, execute [RCL] *nn* to recall the flag data, enter 36.41 to specify flags 36 through 41 (the display format flags), and then execute [STOFLAG] to restore flags 36 through 41 to their previous status.

This allows input and output to occur in any display format, while the [FIX] 0 format is in effect as required.

Multiple Copies of Flags. You can use [RCLFLAG] and [STOFLAG] rather than [X<>F] to handle multiple copies of flags, with the following differences.

- Each flag group can be any block of the user flags, including the entire block of flags 00 through 10. (For [X<>F] each flag group must be flags 00 through 07.)
- You can't use the flag data obtained by [RCLFLAG] except to restore the actual flags to their previous status. (The last example for using [X<>F] illustrated a way to directly use the number obtained by [X<>F]).

Section 20

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.

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 and SAVEP) 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 9). You can then use these keys to execute the corresponding local Alpha labels in the current program.

Program Example. The program Σ in section 22 displays **CLEAR D, J? E**, which is a menu offering three alternatives. You can press [D] to clear days, press [J] to clear a job, or press [E] to exit. When you press one of these keys, program execution starts at the corresponding local Alpha label.

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.

Program Example. To conserve program memory, the program Σ in section 22 uses short-form local numeric labels wherever possible. Because there are 15 short-form labels and 28 places to use them, Σ uses short-form labels 00 through 03 more than once. This is made possible by restricting the use of labels 00 through 03 to forward branches (where the [GTO] instruction precedes the [LBL] instruction). Although LBL 00 appears in five distinct program lines, each GTO 00 is intended for the next LBL 00 in the program and so no confusion occurs.

In contrast, short-form labels 04 through 14 are used for reverse branches (where the [GTO] instruction follows the [LBL] instruction). Because the computer must search from the [GTO] instruction to the bottom of the program and then from the top of the program to the [LBL] instruction, each of these labels appears only once to avoid confusion.

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.

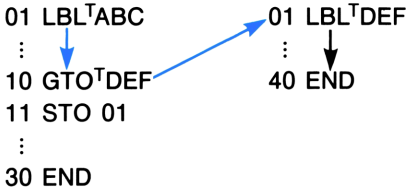
Calling a Subroutine

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:

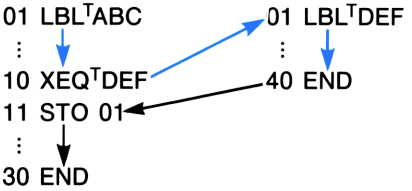
- Both transfer program execution to the specified label.
- All types of labels that can be specified for GTO can also be specified for XEQ.

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.

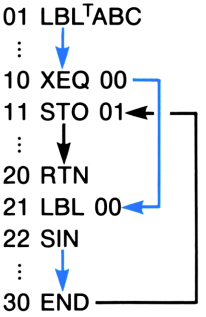


Program ABC calls program DEF as a subroutine, so execution returns to ABC when the END instruction is encountered at the end of DEF.



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.)



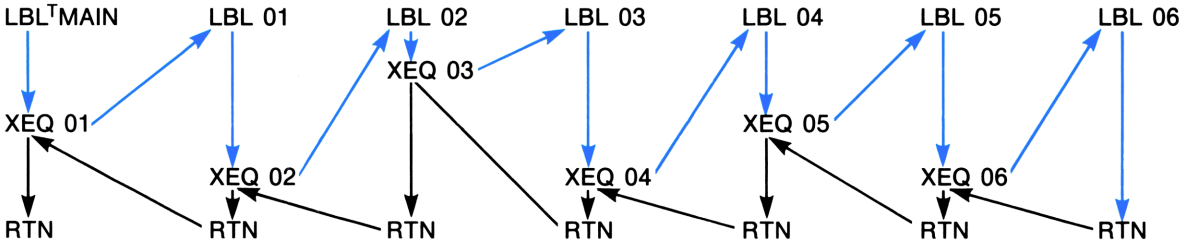
If you call ABC as a subroutine from another program, execution returns to the calling program when 20 RTN is executed. That is, if a program calls a subroutine that calls a second subroutine, the second subroutine is completed and execution returns to the first subroutine; then the first subroutine 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. This rule also applies if an alarm executes DEF while ABC is stopped. Whether executed by an alarm or by you, 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 H 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

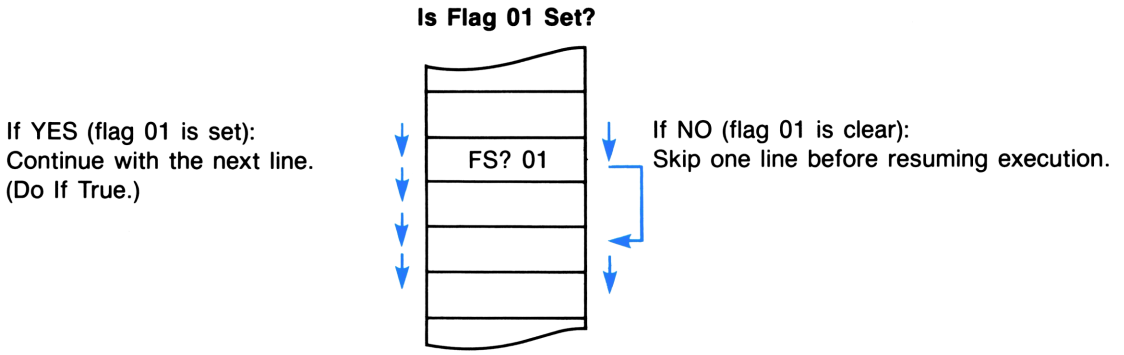
The number of bytes of program memory required by an XEQ instruction depends on which type of label is specified.

- 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.



Flag Tests

The following functions can test any flag.

FS? *nn* Is flag *nn* set? ($00 \leqslant nn \leqslant 55$)

FC? *nn* Is flag *nn* clear? ($00 \leqslant nn \leqslant 55$)

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*. ($00 \leqslant nn \leqslant 29$)

FC?C *nn* Is flag *nn* clear? Clear flag *nn*. ($00 \leqslant nn \leqslant 29$)

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.

Comparing X with Indirect Y. There are six functions that compare the contents of the X-register with the contents of a register specified in the Y-register. The specified register can be any main memory register (R_{00} through $R_{(319)}$) or the stack or LAST X registers. Place the address from 00 through 319 or the single letter X, Y, Z, T, or L in the Y-register. The functions are:

[X < NN?] [X <= NN?] [X = NN?] [X ≠ NN?] [X >= NN?] [X > NN?]

These functions can compare any combination of Alpha and numeric data. Alpha strings are compared on the basis of character codes, allowing Alpha data to be alphabetized. Alpha data are considered to be strictly greater than numeric data.

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 T LOOP
02 BEEP
03 GTO T 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 LBL T ABC
02 1
03 −
04 X>0?
05 GTO T ABC
06 BEEP
07 END
```

Program Example. The sample programs TR and Σ in section 22 use loops with conditional functions to access extended memory. When many records in the file TRECS need to be processed in a certain way but the number of records in the file varies, the loop that processes each record uses the Error Ignore flag (flag 25) to exit when all records have been processed.

- After each record is processed, flag 25 is set. This means that when the next error occurs, the computer will clear flag 25 but will *not* halt program execution.
- With flag 25 set, the program attempts to access the next record, and then tests flag 25. If flag 25 has been cleared, this means that there are no further records (an **END OF FL** error occurred) and so the program exits the loop. If flag 25 is still set, this means that the next record does exist and so the loop is repeated to process this record.

One particular example of this technique can be found in the program Σ. Lines 351 through 362 form a loop that accesses each time record. Within this loop, lines 354 through 360 attempt to set the pointer to the next time record; if [SEEKPT] on line 358 causes an error, [GTO] 12 on line 360 terminates the loop.

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.

Program Example. The program Σ in section 22 uses loops with loop-control functions to access a block of sequential data registers. When the registers are filled with data recalled from extended memory, they are accessed in order of increasing addresses. When the data are displayed, the registers are accessed in order of decreasing addresses. (The order of the data must be reversed to display the data chronologically).

To fill the registers in order of increasing addresses, Σ uses ISG (on line 213) to increment the loop-control number (in R_{05}). To display the data in the reverse order, Σ uses DSE (on lines 231 and 259) to decrement the loop-control number.

Σ defines the initial and final counter values to be the addresses of the first and last registers in the block. Then each time the loop is executed, the integer part of the loop-control number is the address of the register to access. This use of the loop-control number for indirect addressing (on lines 209, 212, 228, and 254) depends on the following initial value for the loop-control number:

- *iiiii* is equal to the address of the first register to be indirectly addressed.
- *fff* is equal to the address of the last register to be indirectly addressed.
- No value is specified for *cc*, so a default value of 01 will be used.

Section 21

Alpha and Interactive Operations

Contents

Introduction

This section covers two aspects of the Alpha register: advanced manipulations of data in the Alpha register, and the interaction between the user and a program.

- Advanced Alpha register manipulations enhance the standard uses of the Alpha register (such as messages to be displayed and data for text files), and also provide additional characters and the ability to store arbitrary bytes of data.
- Interaction between the user and a program involves the functions that display a message and the functions that interpret the user's response.

The Alpha and X-Registers

There are three ways to move data between the Alpha register and the X-register:

1. 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 12, "Main Memory."
2. Executing [XTOA] *(X to Alpha)* translates a number in the X-register into a character in the Alpha register, and executing [ATOX] *(Alpha to X)* translates the Alpha character back into its decimal equivalent in the X-register.
3. Executing [ANUM] *(Alpha number)* searches the Alpha register for a string of digits and returns the string to the X-register as a number. This is the *only* way to retrieve digits from the Alpha register in a form usable in computations.

Translating Characters and Numbers

The Alpha register can hold a greater variety of characters than can be displayed by the computer, and the display can show a greater variety of characters than you can key in from the Alpha keyboard. The functions [ATOX] and [XTOA] enable you to use the full capabilities of the Alpha register and the display.

Numbers and Characters as Bytes. Recall from section 19 that a byte corresponds both to the status of flags 00 through 07 and to a number from 0 through 255. A byte also corresponds to a character in the Alpha register. Although some bytes cannot be distinguished as characters in the display, they are distinct in the Alpha register itself. The ability of the computer to work with bytes is particularly valuable for controlling peripheral devices.

The null byte 00000000, which corresponds to the decimal value 0, has a special meaning in the Alpha register. Because of this, under some circumstances you can't retrieve a null byte from the Alpha register. These restrictions are discussed in appendix C, "Null Characters." All other bytes can be freely stored, manipulated, and recalled from the Alpha register.

Using [XTOA]. With a number from 0 through 255 in the X-register, execute [XTOA] to append the corresponding byte to the right-hand end of the Alpha register. The X-register is unchanged. If the X-register contains an Alpha string, executing [XTOA] appends that string to the Alpha register. Note that a string of Alpha digits in the X-register is appended as those digits, not as the corresponding byte.

The following table shows all the characters that can be displayed, some of which are not available from the Alpha keyboard. To append any of these characters to the Alpha register, place the decimal value in the X-register and execute [XTOA]. All decimal codes from 128 through 255 produce the "starburst" display (all display elements lit). The table also shows the ASCII display characters (but not control characters) for the corresponding decimal values.

Character Codes

Code	ASCII	Display	Code	ASCII	Display	Code	ASCII	Display	Code	ASCII	Display
0			32	space		64	@		96	`	
1			33	!		65	A		97	a	
2			34	"		66	B		98	b	
3			35	#		67	C		99	c	
4			36	$		68	D		100	d	
5			37	%		69	E		101	e	
6			38	&		70	F		102	f	
7			39	'		71	G		103	g	
8			40	(		72	H		104	h	
9			41	)		73	I		105	i	
10			42	*		74	J		106	j	
11			43	+		75	K		107	k	
12			44	,		76	L		108	l	
13			45	-		77	M		109	m	
14			46	.		78	N		110	n	
15			47	/		79	O		111	o	
16			48	0		80	P		112	p	
17			49	1		81	Q		113	q	
18			50	2		82	R		114	r	
19			51	3		83	S		115	s	
20			52	4		84	T		116	t	
21			53	5		85	U		117	u	
22			54	6		86	V		118	v	
23			55	7		87	W		119	w	
24			56	8		88	X		120	x	
25			57	9		89	Y		121	y	
26			58	:		90	Z		122	z	
27			59	;		91	[		123	{	
28			60	<		92	\		124	|	
29			61	=		93	]		125	}	
30			62	>		94	^		126	~	
31			63	?		95	_		127		

Using [ATOX]**.** When you execute [ATOX], the decimal value of the leftmost byte in the Alpha register is placed into the X-register. This byte is then lost as the contents of the Alpha register shift left, and the stack lifts unless stack lift is disabled. If the Alpha register is empty, executing [ATOX] returns a value of zero.

Any null bytes at the left-hand end of the Alpha register are ignored by [ATOX]. A null byte embedded in an Alpha string will effectively disappear when shifted or rotated into the leftmost position. However, you can detect when a null byte disappears by comparing the length of the Alpha string before and after shifting or rotating the Alpha register, as described in "Finding the Length of a String" below.

Program Example. The programs TR and Σ in section 22 use [XTOA] and [ATOX] to encode the number of hours worked each day on a job. Bytes with decimal values from 1 through 240 represent from .1 through 24.0 hours. This is a very compact way to store data; seven bytes, representing a week's data, consume one extended-memory register in the text file TRECS. The alternative of storing each day's hours in one register of a data file would offer greater accuracy but would consume seven times the memory.

Retrieving a Number from the Alpha Register

The [ANUM] function scans the Alpha register for Alpha digits. If a string of digits is found, it is placed in the X-register and flag 22 (Numeric Data Input) is set. The result in the X-register is a number usable for calculations. If no digits are found, the X-register and flag 22 are unchanged.

If you place two numbers in the Alpha register without separating them, [ANUM] might not distinguish them. For arbitrary characters in the Alpha register, [ANUM] first identifies and then evaluates the digit string.

The string is identified as follows:

- [ANUM] searches from the left end of the Alpha register.
- Characters are ignored until a digit is encountered. If there are immediately preceding radix marks (as currently defined by flag 28) or minus signs, they are considered as part of the string.
- All subsequent minus signs, radix marks, and up to 10 digits are considered as part of the string.
- **E**'s (for "exponent") are considered as part of the string only if there are subsequent digits in the string.
- The separator mark (as currently defined by flag 28) is considered as part of the string only if flag 29 (digit grouping) is set.
- [ANUM] considers the string to be complete when any character not considered as part of the string is encountered.
- [ANUM] stops searching when it encounters the null byte if digits have been encountered.

When a string is identified, ANUM evaluates it as follows:

- A simple string of digits, for instance **12345**, is simply that number, 12345.
- The first **E** encountered causes the subsequent (one or two) digits to be considered as an exponent. All subsequent **E**'s are ignored. If there are already more than eight digits in the string, an **E** is ignored unless there is a radix mark preceding the eighth digit.
- The resulting number in the X-register is positive if there are an even number of minus signs in the string and negative if there are an odd number of minus signs. If an **E** defines an exponent, this rule applies separately to the exponent string.
- The first radix mark encountered divides the integal and fractional parts of the number in the X-register. All subsequent radix marks are ignored. All radix marks after an **E** are ignored.
- All separator marks are ignored. (Separator marks are considered as part of the string—and ignored—only if flag 29 is set; if flag 29 is clear, ANUM considers the string complete if a separator mark is encountered.)

Manipulating Alpha Strings

There are many sources for the characters in the Alpha register. The information might come from your own text files, from an HP-IL peripheral device, or from the Alpha keyboard. Whatever the source, you can combine the fundamental operations described below to perform complex manipulations on that information.

Searching and Rotating the Alpha Register

Recall that the three functions that move data from the Alpha register to the X-register all work from the left end of the Alpha register:

- ASTO X copies the six leftmost bytes into the X-register.
- ATOX translates the leftmost byte into a number in the X-register.
- ANUM identifies the leftmost complete digit string and translates it into a number in the X-register.

To use all the data in the Alpha register, you can rotate the contents to place a specific byte or string at the left end. To do so, first locate the desired data using POSA *(position in Alpha)* and then relocate that data to the left end using AROT *(Alpha rotate)*.

Searching the Alpha Register. POSA searches the Alpha register for a target specified in the X-register. A number representing the position of the target is returned to the X-register; the target specification is saved in the LAST X register. The Alpha register is unchanged.

The target can be:

- The decimal value of a byte (0 through 255); or
- An Alpha string placed in the X-register by ASTO X.

Positions in the Alpha register are counted from left to right, starting with position 0. The position of an Alpha string is defined as the position of the first character in the string. If the target is found, the number representing the target's position replaces the target specification in the X-register. If the target is not found, a value of −1 replaces the target specification in the X-register.

Executing POSA finds only the first occurrence of a target. If the target is the null byte (decimal value 0), only a null byte that follows a non-null byte will be found.

Rotating the Alpha Register. Executing AROT rotates the contents of the Alpha register by the number of positions given in the X-register. Rotation is to the left if the number in the X-register is positive, or to the right if the number is negative. Only the current contents are rotated, not all 24 positions in the Alpha register. Note that executing AROT immediately after POSA rotates the target to the left end of the Alpha register (assuming the target was found).

Any null bytes that are rotated to the left end of the Alpha register effectively disappear. You can check if null bytes have disappeared by keeping track of the length of the string in the Alpha register.

Finding the Length of a String

Executing ALENG returns the number of characters in the Alpha register to the X-register. The Alpha register is unchanged, and the stack lifts unless stack lift is disabled. The following situations suggest ways to use ALENG in conjunction with other Alpha manipulations:

- If you are rotating the Alpha register several times to retrieve numbers with ANUM, you can use ALENG to determine when you've completed one full rotation. When the total number of positions rotated equals the number of characters, the contents have returned to their original order.
- If you are moving bytes to the X-register with ATOX and processing them with a subroutine, you can use ALENG first to determine the required number of iterations.
- If you shift or rotate Alpha strings that contain embedded null bytes, you can use ALENG to detect how many (if any) null bytes disappear. Compare the length of the string before and after you shift or rotate the string; if there are fewer characters afterwards, one or more null bytes have disappeared.

Example of Alpha Manipulations

Suppose that a peripheral device places two digit strings, separated by a space, into the Alpha register. This example shows how to retrieve the two numbers, placing the first digit string in the X-register and the second in the Y-register. The comments on the right assume a combined digit string of "**123.45 678.90**" in the Alpha register.

01 ANUM	The first digit string, 123.45, is placed in the X-register. The contents of the Alpha register are unchanged.
02 32	The decimal value for a space, 32, is placed in the X-register to be the operand for the next two steps. 123.45 is lifted into the Y-register.
03 XTOA	A space is appended to the end of the second digit string. This ensures that the two digit strings won't be joined when the contents rotate. The Alpha register now contains "**123.45 678.90** ".
04 POSA	The number 6, the position of the space separating the two digit strings, replaces 32 in the X-register.
05 AROT	The contents of the Alpha register rotate 6 positions to the left. The Alpha register now contains " **678.90 123.45**".
06 ANUM	The second digit string, 678.90, is placed in the X-register. 6 is lifted into the Y-register and 123.45 is lifted into the Z-register.
07 RCL Z	The number 123.45 is recalled to the X-register and 678.90 is lifted into the Y-register.

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.

Program Examples. The programs TR and Σ in section 22 use [PROMPT] for a variety of purposes. The most common purpose is to ask you to confirm a displayed number (time or date) or Alpha string (job name). You can respond in two ways: either key in an alternative value and press [R/S], or simply press [R/S] to confirm the displayed value. The programs use flags 22 and 23 to determine whether you keyed in an alternative value.

- If you key a number into the X-register, flag 22 (Numeric Data Input) is set. This flag can be tested later to check whether a number was entered.
- If you key an Alpha string into the Alpha register, flag 23 (Alpha Data Input) is set. This flag can be tested later to check whether an Alpha string was entered.

For example, when TR asks which job you're starting (lines 135 through 146), it clears flag 23 before prompting and later tests whether flag 23 was set. In addition, the program executes [AON] *(Alpha on)* before [PROMPT], and [AOFF] *(Alpha off)* after [PROMPT], so that the Alpha keyboard will be active while execution is halted.

When TR attempts to enlarge the text file TRECS but there are no extended memory registers available, it uses [PROMPT] to display **NEED ROOM** and halt execution (lines 230 and 231). Your response should be a procedure, namely reducing the size of some other file. After you do so and press [R/S], TR tries again to enlarge TRECS.

The program Σ uses [PROMPT] to display **CLEAR D, J? E** and halt execution (lines 303, 304, and 305). When you press [D], [J], or [E], execution resumes at the corresponding local Alpha label (LBL D on line 341, LBL J on line 308, or LBL E on line 363).

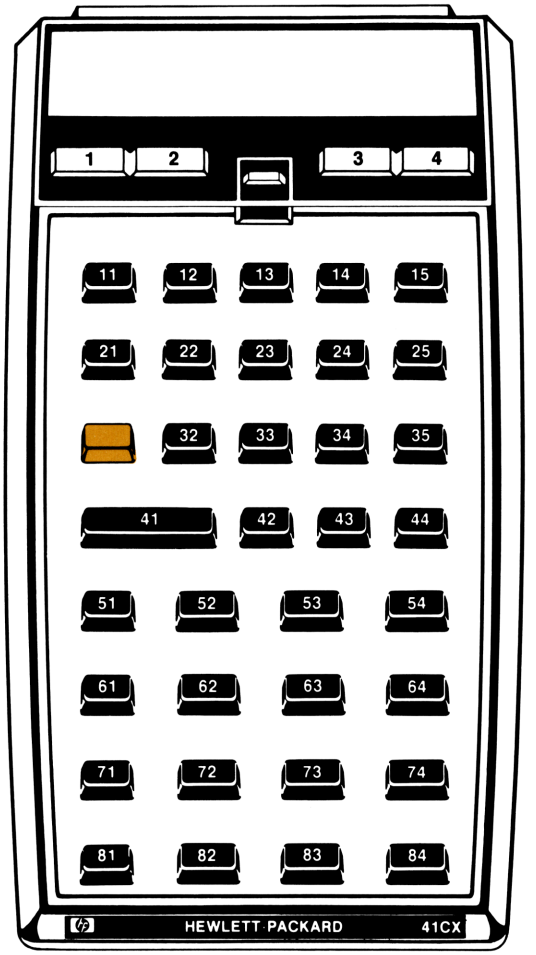
Using [PSE]

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.

Keycodes for GETKEY and GETKEYX



GETKEY returns a positive keycode to the *X-register*. Pressing [f] returns a keycode of 31.

GETKEYX returns a positive *or negative* keycode to the *Y-register*. Pressing [f] first returns a negative keycode for the second key pressed.

Responding to a Pressed Key

Two functions, [GETKEY] and [GETKEYX], wait for you to press a key and then return a keycode representing the key you pressed. The diagram on the facing page shows the keycode for each key.

Using [GETKEY]

Executing [GETKEY] delays program execution up to 10 seconds. If you press a key within 10 seconds, [GETKEY] returns the appropriate keycode to the X-register, lifting the stack unless stack lift is disabled. If you don't press a key, [GETKEY] returns zero. To display a message while waiting, a program can place the message in the Alpha register and execute [AVIEW] before executing [GETKEY].

Note that all values returned by [GETKEY] are valid values for local numeric labels. You can use [GETKEY] and [XEQ] IND X to branch to the subroutine appropriate to the user's response. This technique is somewhat like local Alpha labels, but it doesn't clear the subroutine return stack. First associate each response with a letter and find the keycode of that letter's key. Then start each response's subroutine with a numeric label that is the keycode for that letter's key. If you want the user to press [Y] for "yes" or [N] for "no," start the subroutine for "yes" with [LBL] 71 and the subroutine for "no" with [LBL] 41. Then follow [GETKEY] with [XEQ] IND X. When [GETKEY] is executed and the user presses [Y] or [N], either 71 or 41 is returned to the X-register and [XEQ] IND X will execute the appropriate subroutine.

Using [GETKEYX]

[GETKEYX] is similar to [GETKEY] but includes the following features:

Variable Interval. A number $SS.s$ in the X-register, $|SS.s| \leq 99.9$, specifies how many seconds [GETKEYX] waits for a response.

Shifted and Unshifted Keycodes. [GETKEYX] returns a keycode to the Y-register. Pressing [f] followed by another key produces a negative value of the second key's keycode. Pressing [f] restarts the specified interval; if you don't press a second key within the second interval, [GETKEYX] returns zero indicating that no key was pressed.

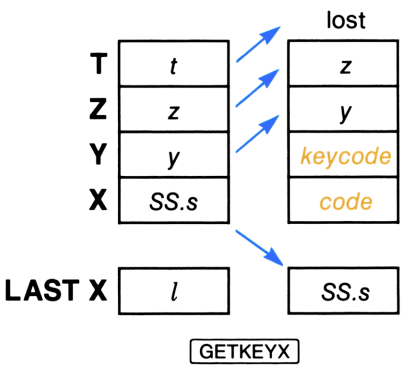
Character Codes. [GETKEYX] returns a character code (if appropriate) to the X-register, according to the table on page 310. If flag 48 is set (the Alpha keyboard is active), a character code is returned for keys with characters on the Alpha keyboard. If flag 48 is clear, a character code is returned for the digit keys, the radix-mark key, and [CHS]. For all other cases [GETKEYX] returns zero to the X-register. Note that a key has different character codes depending on flag 48: keycode 52 corresponds to the letter R (character code 82) on the Alpha keyboard, and to the digit 7 (character code 55) on the Normal keyboard.

Action on Key Up or Key Down. The sign of the number *SS.s* in the X-register determines whether the computer will act when the key is pressed or when the key is released:

- If $SS.s \geq 0$, the computer acts when the key is released. This is the simpler option; the function on the key is never executed and there will be one response each time a key is pressed.
- If $SS.s < 0$, the computer acts when the key is pressed down, making repeating keys possible. For example, a program can contain a loop that places −.01 in the X-register, executes GETKEYX, calls a subroutine according to the user's response, and then starts over. Executing GETKEYX with a small negative value for SS.s asks, Is there a key down now? If the user holds down a key, the loop containing GETKEYX would call the appropriate subroutine over and over until the key is released.

  Releasing the key produces the normal effect of pressing the key; but remember that only R/S and ON have any effect while a program is running.

GETKEYX lifts the contents of the Y- and Z-registers into the Z- and T-registers, and moves the interval *SS.s* from the X-register to the LAST X register, as illustrated below.



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 D.

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 22

Programs for Keeping Time Records

Contents

Introduction

This section contains programs that can help you keep track of how you spend your time. Instructions appear below under "Using the Programs" and are followed by an explanation of each program. When you understand what these programs do, you can study the annotated program listings to understand how they work.

Although you may never need to write programs as involved as these, you may find some of the techniques useful in your own applications. Each program is divided into major parts that correspond to major aspects of the program's operation. Each major part is then divided into routines—sequences of program lines that perform a basic task. You can learn a great deal about the individual functions in a routine by observing how they work together to perform the task, and you may find some of these routines useful in your own programs.

Program Examples

Sections 19, 20, and 21 refer to these programs to illustrate certain programming functions and techniques. The topics discussed in these "Program Examples" include:

- Data Input and Error Ignore flags ("Control Flags" in section 19).
- Local Alpha labels and local label searches ("Local Labels" in section 20).
- Looping using conditional functions and using loop-control functions ("Looping" in section 20).
- Efficient storage of data ("Translating Characters and Numbers" in section 21).
- Program/user interaction ("Using PROMPT" in section 21).

Using the Programs

There are four processes involved in keeping track of your time:

- Loading the programs TR *(time records)* and Σ *(summary)*.
- Creating the files that will hold the information. You will need to do this only once, unless you later clear or purge these files or reset the computer.
- Updating the information. You will need to do this each time you start or stop working on a job.
- Summarizing the information. You can do this whenever you want a record of how you've spent your time.

The steps you should follow for each process are given below.

Loading the Programs

This section contains listings for the programs TR and Σ. To key in the programs, follow the instructions under "Loading a Program" in section 18.

Bar code for the programs appear in appendix J, "Bar Code for Programs." To enter the programs using an HP 82153 Wand, follow the instructions in the owner's manual for the wand.

Creating the Files

Creating the File LAST. This data file will hold three values describing the situation when you last started or stopped working: *last job*, the name of the job you started or stopped; *last time*, the time of day; and *last date*. To begin keeping time records you need to create this file and give initial values for the time and date.

1. Key **LAST** into the Alpha register.
2. Key 3 into the X-register.
3. Execute CRFLD. This creates a data file named LAST that contains three registers. The pointer is set to the first register.

4. Key 1 into the X-register.
5. Execute [STO] 02. The initial value for *last time* will be 1.
6. Execute [DATE]. This returns the current date to the X-register. (The contents of the X-register differ from the resulting display.)
7. Execute [STO] 03. The initial value for *last date* will be the current date.
8. Key 1.003 into the X-register.
9. Execute [SAVERX]. This copies the contents of R_{01}, R_{02}, and R_{03} into LAST. (The initial value for *last job* will be the current contents of R_{01}. You will replace this with a real name when you first execute TR.)

Creating the File TRECS *(time records).* This text file will contain a pair of records for each job that you work on. To begin keeping time records you need only to create this file; the updating program TR will automatically create the necessary records within TRECS.

1. Key **TRECS** into the Alpha register.
2. Calculate $j\ (w + 1)$, where j is the number of jobs and w is the number of weeks for which you want to keep records. This is an estimate of the number of registers TRECS will require; it can be a rough estimate because the program TR will automatically add registers to TRECS if necessary. Key this estimate into the X-register.
3. Execute [CRFLAS]. This creates the text file TRECS containing the estimated number of registers.

Creating the First Records. Next you need to execute TR once to create a pair of records in TRECS for your first job. (One record is for the job's name; the other record is the job's time record.) Unless you're really starting work on that job now, you'll want to repeat TR immediately to indicate that you're stopping. No elapsed time will be recorded if you stop within three minutes of when you started.

1. Execute TR.
2. When the computer displays **JOB NAME = ?**, key in the name of your first job and press [R/S]. (You don't need to press [ALPHA] before or after keying in the name.)
3. When the computer displays **START *job*?**, where *job* is the job name you just keyed in, press [R/S] to confirm.
4. When the computer asks **NEW JOB?**, press [R/S]. This confirms that the name you keyed in isn't a misspelled existing name. The computer now creates the pair of new records for your job.
5. When the computer asks **START NOW?**, press [R/S] to confirm.

When zero returns to the display, you have successfully completed your first execution of TR. You can indicate that you're stopping now by repeating TR.

6. Press [R/S] to start TR again. (Executing TR twice in a row is common when you stop work on one job and immediately start on another.)
7. When the computer displays **STOP NOW?**, press [R/S] to confirm.

When zero returns to the display, you're ready to use TR according to the rules for general use that follow.

Using TR

Execute the program TR each time that you start or stop work on a job. The program will offer you different choices depending on whether you're starting or stopping; these choices are described below under "Starting Work" and "Stopping Work." The following instructions apply to both situations.

General Instructions.

- There are three variables that appear in the instructions for starting and stopping:

  Last job is the name of the job that you started or stopped when you last executed TR. This value changes only when you key in a different job name.

  Last time is the time of day that you started or stopped most recently. This value changes each time you execute TR. (The time you start is stored as a negative number; a value of 1 is stored when you stop. When you next execute TR, the sign of the stored value of *last time* indicates whether you started last time and so are stopping now, or vice versa.)

  Last date is the date that you started or stopped most recently. This value is handled automatically by TR.

- When the computer displays a question, it is usually indicating a default value for a name or time. To confirm the displayed value, just press [R/S]. To select an alternative value, key in that value before you press [R/S].

- To key in the time of day, you can use either a 24-hour format:

  $$HH.MM = HH \text{ hours } (0 \leqslant HH < 24) \text{ and } MM \text{ minutes } (0 \leqslant MM < 60)$$

  or else you can enter a time between noon and midnight as a negative number:

  $$-HH.MM = 12 + HH \text{ hours } (0 \leqslant HH < 12) \text{ and } MM \text{ minutes } (0 \leqslant MM < 60)$$

- If the computer displays **JOB NAME = ?**, key in a job name and press [R/S]. (You don't need to press [ALPHA].) The computer now assumes that you're starting work on that job. This will occur only the first time you execute TR or if you just cleared the pair of records in TRECS for *last job*.

- If the computer displays **NEED ROOM**, reduce the size of an extended memory file other than LAST or TRECS by two or more registers, and then press [R/S]. This will occur when the computer it is unable to enlarge TRECS because there are no registers available in extended memory.

Starting Work.

1. Execute TR.
2. When the computer displays **START *last job*?**, you can either:
   - Press [R/S] to confirm that you're starting again on the same job as last time. The computer then skips to step 4.
   - Key in an alternative job name and press [R/S]. (You don't need to press [ALPHA].) If you key in the name of an existing job—one that you've already used with TR—the computer skips to step 4.
3. If the computer doesn't recognize the job name you keyed in, it displays **NEW JOB?** You can then either:
   - Press [R/S] to confirm that you keyed in the name of a new job. The computer will create a new pair of records in TRECS for this new job.
   - Key in an existing job name and press [R/S]. (You don't need to press [ALPHA].) If you misspelled the job name in step 2, this is your chance to correct your error.
4. When the computer displays **START NOW?** you can either:
   - Press [R/S] to confirm that you're starting now.
   - Key an alternative starting time into the X-register and press [R/S].

The program is done when the display shows zero.

Stopping Work.

1. Execute TR.
2. a. If today's date is the same as *last date*, the computer displays **STOP NOW?** You can then either:
      - Press [R/S] to confirm that you're stopping work now.
      - Key an alternative stopping time into the X-register and press [R/S].

   b. If today's date differs from *last date*, the computer displays **PAST 24:00?** You can then either:
      - Press [R/S] to confirm that you worked past midnight. The computer then updates the job's time record for the day you started (from *last time* through midnight); changes the values for *last time* and *last date* to indicate that you started work at 00:00 hours on the next day; and then goes back to step 2a or 2b.
      - Key an alternative stopping time into the X-register and press [R/S]. (You would do this if you stopped work the same day you started, but forgot to execute TR.)
3. If you keyed in a stopping time that is earlier than the starting time, the computer displays **START > STOP** and goes back to step 2a or 2b.

The program is done when the display shows zero. If you're starting on a different job now, you can press [R/S] to execute TR again.

Using Σ

The program Σ has two stages, summarizing and clearing. The computer first asks for the dates you want summarized and produces the summary; then you are offered the options of clearing time records for all jobs before a specified date or else clearing an entire job.

You can summarize any period for which time records exist, including parts of the entire period. For example, suppose that you accumulate time records for a month. During that month you could summarize each week on Monday of the following week. At the start of the following month you could then summarize the past month, clear the time records for that month, and start the cycle again.

Instructions for the two stages appear under "Summarizing Time Records" and "Clearing Time Records." The following instructions apply to both stages.

General Instructions.

- There are two variables that appear in the instructions for summarizing and clearing.

  Earliest date is the date of the oldest time record to be summarized or cleared. The default value is the earliest date for which time records exist. When you're summarizing you can specify a later date (as long as it isn't later than *most recent date*).

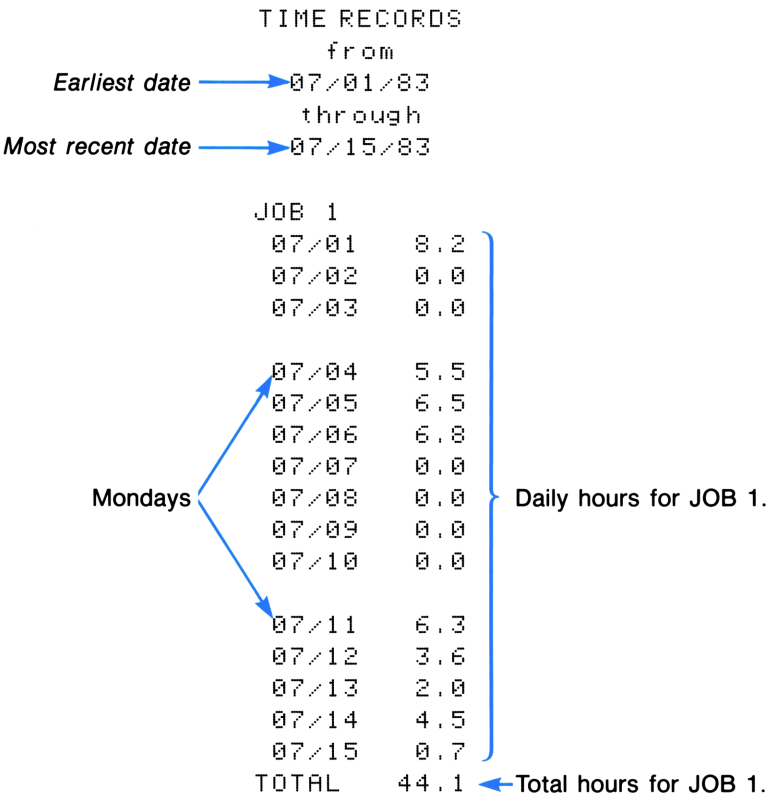
  Most recent date is the date of the latest time record to be summarized or cleared. The default value is the most recent date for which time records exist. When you're summarizing or clearing you can specify an earlier date (as long as it isn't earlier than *earliest date*).

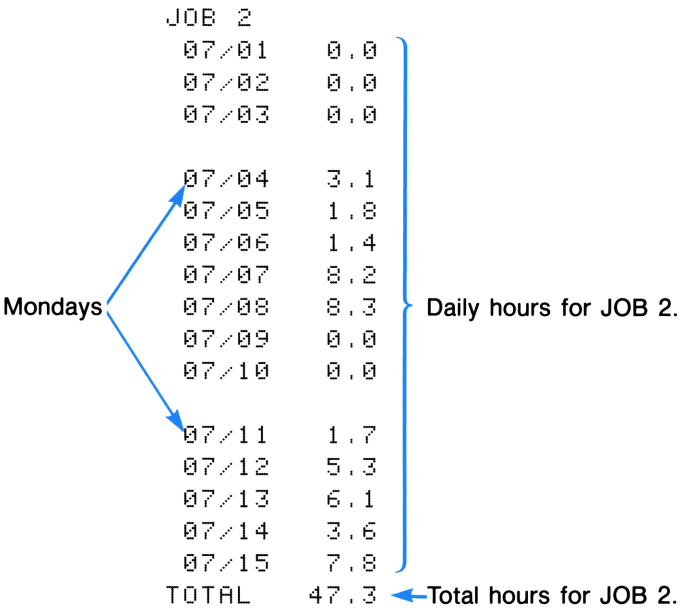
- When the computer displays a question involving a date, it is indicating the current value for *earliest date* or *most recent date*. To confirm the displayed date, just press [R/S]. To select an alternative date, key in that date before you press [R/S].
- If the computer displays **ILLEGAL DATE**, the number you keyed in either represents a date before *earliest date* or after *most recent date*, or else it isn't a valid date at all. In the latter case, be sure you're using the date format ([DMY] or [MDY]) currently in effect.

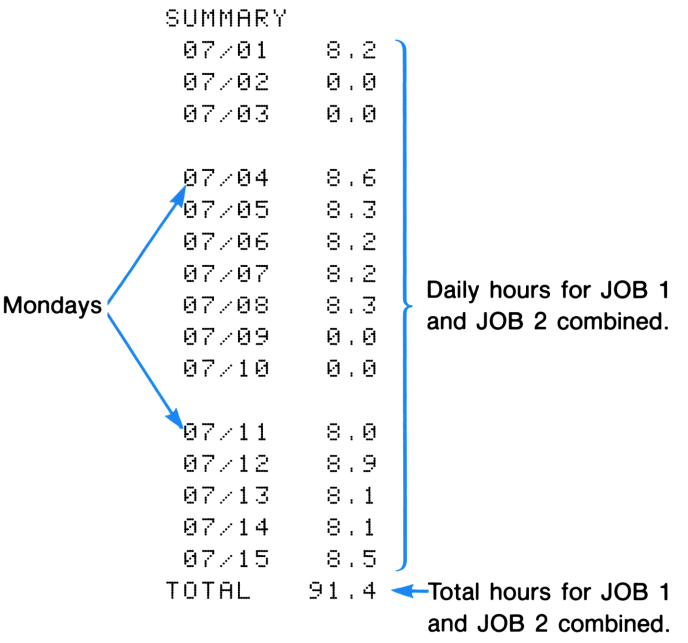
Summarizing Time Records.

1. If you're using an HP 82143A or HP 82162A printer, switch the printer to MAN *(manual mode)*.
2. Execute Σ. (Σ is [f] [F] on the Alpha keyboard.)
3. When the computer displays **FROM *earliest date*?**, you can either:
   - Press [R/S] to confirm this date.
   - Key a later date into the X-register and press [R/S]. The computer will then repeat this step to confirm the later date.
4. When the computer displays **THROUGH *most recent date*?**, you can either:
   - Press [R/S] to confirm this date.
   - Key an earlier date into the X-register and press [R/S]. The computer will then repeat this step to confirm the earlier date.

5. If the computer displays **PACKING** (perhaps very quickly) and **TRY AGAIN**, press [R/S]. This will occur if the program attempted to allocate more main memory registers to data storage than the number of free registers available. If the automatic packing freed enough registers, the program will run smoothly after you press [R/S]; but if there are still too few free registers available, the computer will repeat the message **TRY AGAIN**. In this case you should either:
   - Execute Σ again (starting over with step 1), specifying a shorter summary period than before.
   - Clear a program, alarms, or User keyboard assignments; execute [PACK] or [GTO] [.] [.]; and then execute Σ again (starting over with step 1), specifying the same summary period as before.
6. The computer produces a summary of time records for the period you have specified. The following sample shows a printed summary. If no printer is present, the computer displays each line of the heading for about one second; each subsequent line is displayed until you press [R/S], giving you an opportunity to copy the summary.




JOB 2
07/01 0.0
07/02 0.0
07/03 0.0
07/04 3.1
07/05 1.8
07/06 1.4
07/07 8.2
07/08 8.3
07/09 0.0
07/10 0.0
07/11 1.7
07/12 5.3
07/13 6.1
07/14 3.6
07/15 7.8
TOTAL 47.3
Mondays
Daily hours for JOB 2.
Total hours for JOB 2.


SUMMARY
07/01 8.2
07/02 0.0
07/03 0.0
07/04 8.6
07/05 8.3
07/06 8.2
07/07 8.2
07/08 8.3
07/09 0.0
07/10 0.0
07/11 8.0
07/12 8.9
07/13 8.1
07/14 8.1
07/15 8.5
TOTAL 91.4
Mondays
Daily hours for JOB 1 and JOB 2 combined.
Total hours for JOB 1 and JOB 2 combined.

Clearing Time Records.

7. When the summary is completed, the computer displays **CLEAR D, J? E**. This is a menu that offers three choices; indicate your choice by pressing [D], [J], or [E].

   [D] = Clear days. When the computer displays **THROUGH *most recent date*?** you can either:

   - Press [R/S] to confirm the displayed date. The computer will clear the time records for all jobs for this date and earlier, then return to the menu **CLEAR D, J? E**.
   - Key an earlier date into the X-register and press [R/S]. The computer will ask you to confirm the earlier date.

   [J] = Clear job. When the computer displays **NAME OF JOB?**, key in the name of the job to be cleared and press [R/S]. (You don't need to press [ALPHA].)

   - If the computer finds a job with the name you keyed in, it clears both the name and the time record for that job, then returns to the menu **CLEAR D, J? E**.
   - If the computer doesn't find a job with the name you keyed in, it displays **NO SUCH NAME** and returns to the menu **CLEAR D, J? E**.

   [E] = Exit. This restores the previous flag status (and the options it represents) and then terminates Σ.

8. Press [GTO] [·] [·] or else execute a different program. This is necessary because the [D], [J], and [E] keys will act as described above as long as Σ is the current program and the User keyboard is active.

Files Used To Keep Time Records

Two extended memory files are used to keep time records. One is an text file named TRECS which contains a pair of records for each job. The other is a data file named LAST which contains information from the last time you started or stopped working.

Text File TRECS

TRECS holds the name and time record for each job in a sequential pair of records, as shown in this diagram.

Record Number	Record Contents
000	Name of the first job.
001	Time record for the first job.
002	Name of the second job.
003	Time record for the second job.
⋮	⋮
n	Name of the $n/2 + 1$ job.
$n + 1$	Time record for the $n/2 + 1$ job.
⋮	⋮

Each character in a time record is a byte whose decimal value represents the number of hours you spent on that job for a particular day. The first byte represents the hours for the current day and each subsequent byte represents the hours for each preceding day. This arrangement allows time records to be of different lengths—the nth byte in each time record represents the same day for a different job. The time record for an older job, one for which you have records for a longer period, will be longer than a time record for a newer job. When you create a new job, its pair of records is added to the end of TRECS. This means that the first job in TRECS is the oldest job, and its time record is as long or longer than any other.

The number of hours per day is calculated to one decimal place. This value, multiplied by 10, is the decimal value of the byte that encodes those hours. The following table shows some examples of this system of encoding.

Time (hours/minutes)	Time (decimal hours)	Byte (decimal value)
3 hrs. 20 min.	3.3	33
5 hrs. 30 min.	5.5	55
7 hrs. 0 min.	7.0	70
10 hrs. 42 min.	10.7	107
13 hrs. 15 min.	13.3	133

There are two bytes that are exceptions to this system.

- Because a null byte (decimal value = 0) can disappear in the Alpha register, "no hours" is encoded by the byte having a decimal value of 254.
- The last byte in each time record is an extra "end of time record" byte having a decimal value of 255.

Data File LAST

The second file used to keep time records is a data file named LAST which holds information about the last time that you started or stopped working on a job.

Register Number	Register Contents
000	*Last job* = Name of the last job started or stopped.
001	*Last time* = Time when you last started or stopped.
002	*Last date* = Date when you last started or stopped.

Last job. Job names are limited to six characters because they must fit in a register. If you start and stop work several times on the same project, you won't need to specify its name each time you start because its name is stored in LAST. One job's name shouldn't be identical to the beginning of another job's name. For example, if one job is named ABCD, don't name another job ABC. (However, you can name another job BCD.)

Last time. The updating program TR stores a negative value for *last time* when you've started working and a positive value when you've stopped. This will indicate whether you're starting or stopping work when you next execute TR. If you always remember to execute TR when you actually start or stop work, you'll never need to key in the time.

Last date. When the date stored in LAST differs from the current date, "no hour" bytes are automatically added to the time record for each job. This keeps the time records properly ordered by date.

The program TR copies the contents of these three registers in LAST into R_{01} through R_{03} in main memory whenever you start or stop work. The program uses these values in its operation, updates these values, and then copies the updated values back to LAST.

Explanation of TR

The program TR contains several major parts.

- The main routines recall the contents of LAST from extended memory, evaluate what should be done, execute the appropriate routines, and copy the updated values into LAST in extended memory.
- The routines for stopping work ask when you stopped, calculate the number of hours you worked, update the appropriate time record in TRECS, and update *last time*.

- The routines for a new date add a byte representing "no hours" to each time record and update *last date*.
- The routines for starting work ask which job you're starting, when you're starting, and then update *last job* and *last time*.

In addition there are three minor subroutines at the end of TR to search TRECS for a specified job name, to convert a time value, and to enlarge the file TRECS.

Registers and Flags Used by TR

The program TR uses the following data storage registers and user flags in its operation. Refer to these tables as you read the program listing.

Registers and Their Contents for TR

Register Number	Register Contents
00	Flag status at start.
01	*Last job.*
02	*Last time.*
03	*Last date.*
04	Decimal value of byte representing number of hours worked.

Conditions Represented by User Flags in TR

Flag Number	Condition Represented by Flag
01	Clear = Job name found in TRECS. Set = Job name not found in TRECS.
02	Clear = Stopping work now. Set = Starting work now.
03	Clear = Same date (*last date* = current date). Set = New date (*last date* ≠ current date).

Program Listing for TR

Main Routines. These routines recall data from the last time you executed TR, determine what to do this time, execute the appropriate routines, store the updated data, and conclude the execution of TR.

```
01♦LBL "TR"
02 SIZE?
03 5
04 X>Y?
05 PSIZE
06 RCLFLAG
07 STO 00
08 FIX 1
09 CF 21
10 "LAST"
11 0
12 SEEKPTA
13 1.003
14 GETRX
15 "TRECS"
16 RCLPTA
17 XEQ 50
18 FC? 01
19 GTO 20
20 "JOB NAME = ?"
21 AON
22 PROMPT
23 AOFF
24 ASTO 01
25 1
26 STO 02
```

Lines **01 through 26** recall *last job*, *last time*, and *last date*. Lines **02 through 05** ensure that there are at least five registers allocated to data storage. Lines **06 through 09** save a copy of the current flag status before selecting the display format and clearing the Printer Enable flag. Lines **10 through 14** copy *last job* into R_{01}, *last time* into R_{02}, and *last date* into R_{03}. (These values have been stored in the file LAST since you last executed TR.) Lines **15 and 16** select TRECS as the current file in extended memory. Line **17** calls a subroutine on line 194 that checks if TRECS still contains the job name *last job* and, if so, lines **18 and 19** branch to line 27. If you cleared *last job* from TRECS since you last executed TR, lines **20 through 23** prompt you for a valid job name. Lines 21 and 23 automatically activate and deactivate the Alpha keyboard. Line **24** changes *last name* to the name you keyed in, and lines **25 and 26** set *last time* to indicate that you stopped last time (and so are starting this time).

```
27♦LBL 20
28 CF 02
29 RCL 02
30 X>0?
31 SF 02
32 CF 03
33 RCL 03
34 DATE
35 X≠Y?
36 SF 03
37 FC? 02
38 XEQ 01
39 FS? 03
40 GTO 30
41 FS? 02
42 XEQ 40
43 "LAST"
44 0
45 SEEKPTA
46 1.003
47 SAVERX
48 RCL 00
49 STOFLAG
50 CLX
51 STOP
52 GTO "TR"
```

Lines **27 through 52** direct the remainder of TR's execution. Lines **28 through 31** clear flag 02 if you're stopping work, or else set flag 02 if you're starting work. (If you started work last time, TR stored a negative value for *last time*; if you stopped work last time, TR stored a value of 1 for *last time*.) Lines **32 through 36** clear flag 03 if *last date* is the current date, or else set flag 03 if *last date* differs from the current date. Lines **37 through 42** direct overall execution of the program. If you're stopping work, lines **37 and 38** call the subroutine which begins on line 58. If one or more days have passed since you last executed TR, lines **39 and 40** branch to a routine beginning on line 106 (which adds another byte to each time record and then branches to line 27). If you're starting work, lines **41 and 42** call the subroutine that begins on line 135. Lines **43 through 51** conclude the execution of TR. Lines **43 through 47** copy the current values for *last job*, *last time*, and *last date* into the file LAST, to be saved until you execute TR again. Lines **48 and 49** restore the previous flag status (and the options it represents), and line **50** clears the flag-status data from the X-register. Line **51** stops program execution and line **52** branches to line 01; this enables you to repeat TR simply by pressing R/S.

Stopping Work. These routines update the time record for *last job* when you stop work. Normally only the second and third routines will be executed; the first routine is an error routine, and the fourth routine is executed only when you stop work on a different date from when you started.

```
53♦LBL 00
54 "START > STOP"
55 TONE 5
56 AVIEW
57 PSE

58♦LBL 01
59 FS? 03
60 GTO 03
61 TIME
62 " STOP NOW?"
63 PROMPT
```

Lines **53 through 57** display an error message. If you key in a stopping time that is earlier than the time you started, line 72 branches here to display an error message before beginning again.

Lines **58 through 63** determine the time you stopped work. If you're stopping work on a different day from when you started, lines **59 and 60** branch to line 96. Lines **61, 62, and 63** ask if you're stopping at the current time and then stop execution; if you key in an alternative time, it replaces the current time in the X-register.

```
64♦LBL 02
65 X<0?
66 XEQ 60
67 X=0?
68 24
69 RCL 02
70 HMS+
71 X<0?
72 GTO 00
73 HR
74 RND
75 10
76 *
77 STO 04
78 XEQ 50
79 GETREC
80 ATOX
81 RCL 04
82 +
83 254
84 X≠Y?
85 MOD
86 CLA
87 XTOA
88 RCLPT
89 INT
90 SEEKPT
91 1
92 DELCHR
93 INSCHR
94 STO 02
95 RTN
```

Lines **64 through 95** update the byte for *last date* in *last job*'s time record. If you keyed in a negative time to indicate "p.m.," lines **65 and 66** call a subroutine on line 216 that converts the time to a 24-hour format. If the stopping time is midnight, lines **67 and 68** place 24 in the X-register. Lines **69 and 70** subtract the time you started from the time you stopped: line 69 recalls *last time,* which in this case is the time you started and so is negative; then line 70 adds the time you stopped and (the negative of) the time you started, which calculates the number of hours worked. If the result is negative, lines **71 and 72** branch to an error routine on line 53. Lines **73 through 77** store the number of hours worked as a byte value: lines 73 and 74 convert the number from hours-minutes-seconds format to a decimal number with one decimal place; lines 75 and 76 convert this decimal number to a byte value; and line 77 stores the byte value in R_{04}. Line **78** calls a subroutine on line 194 that locates the time record for *last job*, and then lines **79 and 80** place in the X-register the decimal value of the first byte, which represents the hours worked previously on *last date*. To this byte value lines **81 through 85** add the byte value for hours just worked; lines 83, 84, and 85 adjust the result in case the previous byte value was 254 ("no hours"). Lines **86 and 87** place in the Alpha register the byte whose value is in the X-register. Lines **88, 89, and 90** locate the byte for *last date*, and lines **91, 92, and 93** replace the old byte with the updated byte. Line **94** stores a value of 1 in *last time*. Line **95** returns execution to line 39 if the updating process is complete; if you worked past midnight and this updates only through midnight, execution returns to line 104.

```
96♦LBL 03
97 24
98 " PAST 24:00?"
99 CF 22
100 PROMPT
101 FS? 22
102 GTO 02
103 XEQ 02
104 0
105 STO 02
```

Lines **96 through 105** ask if you worked past midnight. Line 60 branches here if you're stopping work on a different day from when you started. Lines **97 through 100** ask if you worked past midnight; if you key in an alternative time, it replaces 24 in the X-register and lines **101 and 102** branch to line 64. Line **103** calls a subroutine on line 64 that updates the byte for *last date* in *last job*'s time record, from the time you started until midnight; then lines **104 and 105** give *last time* a value of zero to indicate starting at midnight.

Adding Bytes for a New Date. When you execute TR for the first time each day, the program executes these routines to add one byte to the start of each job's time record. This is necessary to maintain the proper correspondence between a byte's position in a time record and the date it represents.

```
106◆LBL 30
107 ASROOM
108 2
109 ∗
110 SF 25
111 SEEKPT
112 FC?C 25
113 GTO 04
114 XEQ 70
115 GTO 30

116◆LBL 04
117 CLA
118 254
119 XTOA
120 1
121 SEEKPT
122 SF 25

123◆LBL 05
124 INSCHR
125 2
126 +
127 SEEKPT
128 FS? 25
129 GTO 05
130 RCL 03
131 1
132 DATE+
133 STO 03
134 GTO 20
```

Lines **106 through 115** ensure that there are enough bytes available in TRECS to add one byte to each time record. Line **107** returns n, the number of available bytes. Suppose that these n bytes were added to the time records for the first n jobs; then the $n + 1$ job is the first that won't receive a byte. Lines **108 and 109** return $2n$, which is the pointer value for the name of the $n + 1$ job. Lines **110 and 111** check if there actually is an $n + 1$ job name; if not, there are enough bytes available and lines **112 and 113** branch to line 116. Line **114** calls a subroutine on line 221 that enlarges TRECS, and line **115** branches to line 106.

Lines **116 through 122** prepare to add one byte to each time record. Lines **117, 118, and 119** place the byte for "no hours" in the Alpha register. Lines **120 and 121** set the pointer to the first character position in the first time record. Line **122** sets the Error Ignore flag; the program will execute the loop that follows until an error clears this flag.

Lines **123 through 134** add one byte to each time record. Line **124** inserts the byte for "no hours" at the current pointer position. Lines **125, 126, and 127** attempt to set the pointer to the first character position in the next time record. If there is another time record, lines **128 and 129** branch to line 123. When all time records have received a byte, lines **130 through 133** add one day to *last date* to reflect the new byte added to each time record. Line **134** branches to line 27. (The updated value of *last date* will be compared to the current date, and the program will execute these routines again if necessary.)

Starting Work. These routines determine which job and what time you're starting work. If you're starting the same job you worked on last time, the first routine is executed. If you key in the name of an existing job—one that you've already used with TR—the second and third routines are also executed. If you key in the name of a new job, the fourth, fifth, and sixth routines are also executed. In all cases the last routine is executed to determine the starting time.

```
135♦LBL 40
136 "START "
137 ARCL 01
138 "├?"
139 CF 23
140 AON
141 PROMPT
142 AOFF
143 FS? 01
144 GTO 07
145 FC? 23
146 GTO 11

147♦LBL 06
148 ASTO 01

149♦LBL 07
150 XEQ 50
151 FC? 01
152 GTO 11

153♦LBL 08
154 CF 23
155 " NEW JOB?"
156 AON
157 PROMPT
158 AOFF
159 FS? 23
160 GTO 06

161♦LBL 09
162 CLA
163 ARCL 01
164 SF 25
165 APPREC
166 FS?C 25
167 GTO 10
168 XEQ 70
169 GTO 09
```

Lines **135 through 146** ask if you're starting the same job that you last worked on. Lines **136 through 141** display **START *last job*?** Lines **140 and 142** automatically activate and deactivate the Alpha keyboard. If you were prompted for a valid job name in the initial routine because the stored value for *last job* wasn't valid, lines **143 and 144** branch to line 149. If you just confirmed that you're starting *last job*, lines **145 and 146** branch to line 183. (If you keyed in an alternative job name now, execution continues with the next routines.)

Lines **147 and 148** make the job name you keyed in the provisional value of *last job*.

Lines **149 through 152** check the alternative job name. Line **150** calls a subroutine on line 194 that clears flag 01 if the job name exists in TRECS. If the name exists, lines **151 and 152** branch to line 183.

Lines **153 through 160** check that you intended to key in the name of a new job. Lines **154 through 157** display **NEW JOB?** Lines **156 and 158** automatically activate and deactivate the Alpha keyboard. If you key in a job name (because you had previously misspelled the name of an existing job), lines **159 and 160** branch to line 147.

Lines **161 through 169** create a record containing the name of the new job. Lines **162 and 163** place the name in the Alpha register. Lines **164 through 167** try to append the name to TRECS and then, if successful, branch to line 170. If there isn't enough room in TRECS to append the name, line **168** calls a subroutine on line 221 that enlarges TRECS, and then line **169** branches to line 161 to try again.

```
170♦LBL 10
171 CLA
172 254
173 XTOA
174 1
175 +
176 XTOA
177 SF 25
178 APPREC
179 FS?C 25
180 GTO 11
181 XEQ 70
182 GTO 10

183♦LBL 11
184 TIME
185 " START NOW?"
186 PROMPT
187 X<0?
188 XEQ 60
189 24
190 MOD
191 CHS
192 STO 02
193 RTN
```

Lines **170 through 182** create a time record for the new job. Lines **171 through 176** place a "no hours" byte and an "end of time record" byte in the Alpha register. Lines **177 through 180** try to append these bytes to TRECS and then, if successful, branch to line 183. If there isn't enough room in TRECS to append the new time record, line **181** calls a subroutine on line 221 that enlarges TRECS, and then line **182** branches to line 170 to try again.

Lines **183 through 193** ask you the starting time. Lines **184, 185, and 186** display **START NOW?**; if you key in an alternative time, that time replaces the current time in the X-register. If you keyed in a negative time to indicate "p.m.," lines **187 and 188** call a subroutine on line 216 that converts the time to 24-hour format. If you keyed in 24 to indicate midnight, lines **189 and 190** change this value to zero. Lines **191 and 192** make the starting time the new value of *last time*, negative to indicate it's a starting time. Line **193** returns execution to line 43.

Searching TRECS for a Job. The program calls this subroutine for two purposes: lines 17 and 150 call it to determine whether a job exists, and line 78 calls it to locate a job known to exist.

```
194♦LBL 50
195 CLA
196 ARCL 01
197 SF 01
198 0
199 SEEKPT
```

Lines **194 through 199** prepare to search TRECS for a job name. Lines **195 and 196** place the job name in the Alpha register. Line **197** sets flag 01 to indicate that the job name hasn't been found; the next routine will clear flag 01 if it finds the name. Lines **198 and 199** set the pointer to the first character in the first job name in TRECS.

```
200♦LBL 12
201 POSFL
202 X<0?
203 RTN
204 ENTER
205 INT
206 1
207 +
208 SEEKPT
209 X<>Y
210 2
211 MOD
212 X≠0?
213 GTO 12
214 CF 01
215 RTN
```

Lines **200 through 215** search TRECS for the job name. Line **201** searches TRECS, starting at the current pointer location. If the job name isn't found, lines **202 and 203** return execution to line 18 or 151 with flag 01 still set. Lines **204 through 208** set the pointer to the first character in the next record, which generally will be the job's time record. However, bytes in a time record might spell out the name by coincidence. To check whether line 201 actually found a job name, lines **209 through 212** check whether the pointer value returned by line 201 is the first character in an even-numbered record. If not, line **213** branches to line 199. Lines **214 and 215** clear flag 01 to indicate that the job name exists, and then return execution to line 18, 79, or 151.

Converting to 24-Hour Format.

```
216♦LBL 60
217 ABS
218 12
219 +
220 RTN
```

Lines **216 through 220** convert a negative number (representing a time after noon) into 24-hour format. The program calls this subroutine on lines 66 and 188.

Enlarging TRECS. The program calls this subroutine when there isn't enough room in TRECS to add a byte to each time record or to append a new record for a new job.

```
221♦LBL 70
222 "TRECS"
223 FLSIZE
224 1
225 +
226 SF 25
227 RESZFL
228 FS?C 25
229 RTN
230 " NEED ROOM"
231 PROMPT
232 GTO 70
233 END
```

Lines **221 through 233** add one register to TRECS. This subroutine is called by lines 114, 168, and 181. Lines **222 through 225** calculate the desired total number of registers. Lines **226 through 229** try to change the size of TRECS to the desired total and then, if successful, return execution to line 115, 169, or 182. If there isn't a register available in extended memory, lines **230 and 231** display an error message and stop execution. After you reduce the size of another file and press R/S, line **232** branches to line 221 to try again.

Explanation of Σ

The program Σ first determines the desired summary period and produces the summary, and then enables you to clear an entire job or clear all time records for a specified period.

Registers and Flags Used by Σ

The program Σ uses the following data storage registers and flags in its operation. Refer to these tables as you read the program listing.

Registers and Their Contents for Σ

Register Number	Register Contents
R_{00}	Previous flag status.
R_{01}	1.*ccc*, where *ccc* = character-pointer value for *most recent date*.
R_{02}	*Most recent date* to be summarized or cleared.
R_{03}	*Earliest date* to be summarized or cleared.
R_{04}	Temporary storage for dates. *nn* = Address of greatest-addressed register used.
R_{05}	Loop-control number.
R_{06}	Total hours for single job.
R_{07}	Total hours for all jobs combined.
R_{08}	Daily hours for most recent date.
⋮	⋮
R_{nn}	Daily hours for earliest date.

Each of the numbers in R_{08} through R_{nn} has the form *xxx.yyy*. The integer part is a byte value representing *xx.x* hours for one job. The fractional part is a byte value representing *yy.y* hours for all jobs combined.

Conditions Represented by User Flags in Σ

Flag Number	Condition Represented by Flag
00	Clear = Prompting for summarizing. Set = Prompting for clearing.
01	Clear = Date keyed in is legal. Set = Date keyed in is illegal.

Program Listing for Σ

```
01♦LBL "SIGMA"
02 RCLFLAG
03 STO 00
04 CF 21
05 1
06 STO 01
07 CF 00
```

Lines **01 through 07** form the initial routine. Lines **02 and 03** store the current flag status in R_{00}. Line **04** disables the printer (if present). Lines **05 and 06** place in R_{01} the pointer value for *first job/most recent date*. Line **07** indicates that the program is summarizing now.

Determining the Dates. These routines determine the period to summarize. If you later choose to clear days, the program uses most of these routines again to determine the period to clear. The status of flag 00 indicates whether the program is summarizing or clearing.

```
08♦LBL 20
09 FIX 4
10 "LAST"
11 2
12 SEEKPTA
13 GETX
14 FC? 00
15 STO 02
16 "TRECS"
17 1
18 SEEKPTA
19 255
20 CLA
21 XTOA
22 CLX
23 POSFL
24 FRC
25 1 E3
26 *
27 −
28 DATE+
29 STO 03
30 STO 04
31 FS? 00
32 GTO 00
```

Lines **08 through 32** calculate the default dates. (When the program is clearing, line 343 calls this part of the program as a subroutine.) Lines **10 through 13** recall *last date*. If the program is summarizing now, lines **14 and 15** make *last date* the default value for *most recent date* in R_{02}; if the program is clearing now, the most recent date summarized remains in R_{02} as the default value. In either case, *last date* remains in the stack to calculate *earliest date*. To make this calculation, lines **16 through 27** locate the "end of time record" byte (decimal value 255) in the first time record and use that character-pointer value to calculate the difference in days between *last date* and *earliest date*. (Remember that the first time record is the oldest time record.) Lines **16 through 26** return the number of bytes that precede the end-of-record byte. When line **27** subtracts this number from 1 (left on the stack from line 17), the result is negative to indicate going backwards in time. Line **28** then calculates *earliest date*. Line **29** makes this date the default value in R_{03}, and line **30** places this date in R_{04} to appear in the first prompt. If the program is clearing, lines **31 and 32** branch to line 45; if the program is summarizing, execution continues to the next routine.

```
33♦LBL 04
34 " FROM"
35 RCL 04
36 XEQ 01
37 FC? 22
38 GTO 00
39 XEQ 02
40 FC? 01
41 GTO 04
42 RCL 03
43 STO 04
44 GTO 04
```

Lines **33 through 44** prompt you for the earliest date to summarize. Lines **34 through 36** ask if the date in R_{04} is the date you want, calling the subroutine on line 62. If you accept the date in R_{04}, lines **37 and 38** branch to line 45. If you key in an alternative date, line **39** calls the subroutine on line 71 to test your date. (This subroutine places your date in R_{04} and then clears flag 01 if your date is legal or else clears flag 01 if your date is illegal.) If your date is legal, lines **40 and 41** branch to line 33 to ask you to confirm your date. If your date is illegal, lines **42, 43, and 44** place the default value in R_{04} again and then branch to line 33.

```
45♦LBL 00
46 RCL 04
47 STO 03
48 RCL 02
49 STO 04
```

Lines **45 through 49** link the prompting routines. Lines **46 and 47** replace the default value for *earliest date* with the date you selected. Lines **48 and 49** place the default value for *most recent date* in R_{04} to appear in the first prompt.

```
50♦LBL 05
51 " THROUGH"
52 RCL 04
53 XEQ 01
54 FC? 22
55 GTO 00
56 XEQ 02
57 FC? 01
58 GTO 05
59 RCL 02
60 STO 04
61 GTO 05
```

Lines **50 through 61** prompt you for the most recent date to summarize or clear. Lines **51 through 53** ask if the date in R_{04} is the date you want, calling the subroutine on line 62. If you accept the date in R_{04}, lines **54 and 55** branch to line 93. If you key in an alternative date, line **56** calls the subroutine on line 71 to test your date. If your date is legal, lines **57 and 58** branch to line 50 to ask you to confirm your date. If your date is illegal, lines **59 through 61** place the default value in R_{04} again and then branch to line 50.

```
62♦LBL 01
63 AVIEW
64 PSE
65 " "
66 ADATE
67 "⊢?"
68 CF 22
69 PROMPT
70 RTN
```

Lines **62 through 70** prompt you for dates. This subroutine is called by lines 36 and 53. Lines **63 and 64** display the contents of the Alpha register (**THROUGH** or **FROM**). Lines **65, 66, and 67** place the date and a question mark in the Alpha register. Line **68** clears the Numeric Input flag (so the program can test whether you keyed in an alternative date). Line **69** displays the contents of the Alpha register and stops execution. Line **70** returns execution to line 37 or 54.

```
71♦LBL 02
72 CF 01
73 STO 04
74 RCL 02
75 SF 25
76 DDAYS
77 FC?C 25
78 GTO 03
79 X<0?
80 GTO 03
81 RCL 03
82 RCL 04
83 DDAYS
84 X<0?
85 GTO 03
86 RTN
```

Lines **71 through 86** test your date. This subroutine is called by lines 39 and 56. Line **72** clears flag 01. (If your date is illegal, the program will branch to line 87 and set flag 01.) Line **73** places your date in R_{04}. Lines **74, 75, and 76** calculate how many days your date is before *last date*. If you keyed in a number that doesn't represent a date, line **76** causes an error and lines **77 and 78** branch to line 87. If your date is later than *last date*, lines **79 and 80** branch to line 87. Lines **81, 82, and 83** calculate how many days your date is after *earliest date*. If your date is earlier than *earliest date*, lines **84 and 85** branch to line 87. If your date passes all three tests, line **86** returns execution to line 40 or to line 57, with flag 01 clear to indicate that your date is legal.

```
87♦LBL 03
88 "ILLEGAL DATE"
89 AVIEW
90 TONE 5
91 SF 01
92 RTN
```

Lines **87 through 92** handle illegal dates. Lines **88, 89, and 90** display an error message and sound a tone. Lines **91 and 92** return execution to line 40 or to line 57, with flag 01 set to indicate that your date is illegal.

```
93♦LBL 00
94 RCL 04
95 RCL 02
96 DDAYS
97 .1
98 %
99 ST+ 01
100 FS? 00
101 RTN
102 RCL 03
103 RCL 04
104 STO 02
105 DDAYS
106 8
107 +
108 STO 04
109 .1
110 %
111 6
112 +
```

Lines **93 through 112** conclude date selection. Lines **94 through 99** update the pointer value in R_{01} for *first job/most recent date*, reflecting your choice of the most recent date to summarize or clear. If the program is clearing now, lines **100 and 101** return execution to line 344. Lines **103 and 104** replace *last date* in R_{02} with your choice of *most recent date*. Lines **102, 103, and 105** calculate the difference in days between the earliest and most recent dates to be summarized; this difference defines the block of registers needed for summarizing. Note that R_{08} is the smallest-addressed register in the block containing daily hours. Lines **106, 107, and 108** place *nnn* in R_{04}, where R_{nnn} is the largest-addressed register in the block. Lines **109 through 112** calculate 6.*nnn*, which represents the block of registers that will contain daily and total hours.

Preparing for the Summary. These routines clear registers and produce the summary heading.

```
113♦LBL 06
114 SF 25
115 CLRGX
116 FS?C 25
117 GTO 01
118 RCL 04
119 1
120 +
121 PSIZE
122 X<>Y
123 GTO 06

124♦LBL 01
125 SF 12
126 CF 13
127 SF 21
128 ADV
129 ADV
130 "TIME RECORDS"
131 SF 25
132 PRA
133 FS?C 25
134 GTO 02
135 CF 21
136 AVIEW
137 PSE
```

Lines **113 through 123** clear the registers for daily and total hours. Lines **114 and 115** attempt to clear the block of registers. If there are enough registers allocated for the entire block—that is, if R_{nnn} exists—lines **116 and 117** branch to line 124. Otherwise, lines **118 through 121** allocate *nnn* + 1 registers, line **122** returns 6.*nnn* to the X-register, and line **123** branches to line 113.

Lines **124 through 137** begin the summary heading. Lines **125 and 126** select double-width and upper-case print modes. Line **127** sets the Printer Enable flag. Lines **128 and 129** advance the printer two lines. Line **130** places the title in the Alpha register. Line **131** sets the Error Ignore flag to prepare for line **132**, PRA *(print Alpha)*, which is a printer instruction. If a printer prints the title, lines **133 and 134** branch to line 138. If no printer is present or if your printer is turned off, line **135** clears the Printer Enable flag (so that AVIEW doesn't stop execution); and lines **136 and 137** display the title.

```
138♦LBL 02
139 SF 13
140 " FROM"
141 XEQ 03
142 " "
143 RCL 03
144 ADATE
145 XEQ 03
146 " THROUGH"
147 XEQ 03
148 " "
149 RCL 02
150 ADATE
151 XEQ 03
152 CF 13
153 SF 21
154 GTO 00
```

Lines **138 through 154** complete the summary heading. Line **139** selects lower-case print mode. Lines **140 through 145** display (print) **from *earliest date***; lines **146 through 151** display (print) **through *most recent date***. Lines 141, 145, 147, and 151 call a subroutine on line 155 that displays (prints) the contents of the Alpha register. Line **152** selects upper-case print mode. Line **153** sets the Printer Enable flag; this ensures that subsequent [AVIEW] instructions will either print the summary entries or else stop execution so that you can write down each displayed entry. Line **154** branches to line 161.

```
155♦LBL 03
156 ADV
157 AVIEW
158 FC? 21
159 PSE
160 RTN
```

Lines **155 through 160** display (print) one line of the heading. This subroutine is called by lines 141, 145, 147, and 151. Line **156** advances the printer (if present). Line **157** displays the contents of the Alpha register; if a printer is present and turned on, the contents of the Alpha register are printed as well. If no printer is present or if your printer is turned off, lines **158 and 159** prolong the display. (Flag 21 was cleared on line 135 if the printer instruction [PRA] caused an error.) Line **160** returns execution to line 142, 146, 148, or 152.

Recalling Time Records. These routines test if a job has data for the summary period and then recall those data from extended memory. The program executes the first short routine only once; the second routine, once for each job; and the last two routines, once for each day for each job.

```
161♦LBL 00
162 FIX 1
163 CLX
164 SEEKPT
```

Lines **161 through 164** prepare to access the first job. Line **162** selects the numeric format for output. Lines **163 and 164** set the pointer in TRECS (the current file) to the first character of the first record, which is the first letter of the first job name.

```
165♦LBL 07
166 RCL 04
167 .1
168 %
169 8
170 +
171 STO 05
172 CLX
173 STO 06
174 GETREC
175 RCLPT
176 INT
177 RCL 01
178 +
179 .001
180 +
181 SF 25
182 SEEKPT
183 FC?C 25
184 GTO 10
185 ADV
186 ADV
187 AVIEW
188 LASTX
189 −
190 SEEKPT
191 GETREC

192♦LBL 08
193 254
194 ATOX
195 X>Y?
196 GTO 02
197 X≠0?
198 GTO 01
199 GETREC
200 GTO 08
```

Lines **165 through 191** prepare to recall one job's time record from extended memory. Lines **166 through 171** place the loop control number 8.*nnn* in R_{05}. Lines **172 and 173** clear R_{06}, which will accumulate the total hours for this job. Line **174** recalls the job's name to the Alpha register. Before displaying (printing) the job's name, the program checks that this job's time record contains data for the summary period. Lines **175 through 178** calculate the pointer value for *most recent date* in this record. (If the record doesn't contain a byte for *most recent date*, it doesn't contain any data for the summary period.) Lines **179 and 180** advance the pointer value one character to ensure that the end-of-time-record byte isn't mistaken for data. Lines **181 through 184** attempt to set the pointer to this pointer value and, if there isn't a byte there, branch to line 243. (If this job is too new for the summary period, all jobs that follow are also too new.) Lines **185, 186, and 187** advance the printer twice and display (print) the job's name. Lines **188, 189, and 190** set the pointer to *most recent date*, and line **191** recalls up to 24 bytes from this job's time record to the Alpha register.

Lines **192 through 200** recall and test each byte. This routine and the following one form a loop. Lines **193 and 194** return 254 to the Y-register and the decimal value of a byte to the X-register. (For sequential executions of this routine, line 194 moves bytes from the Alpha register into the X-register in the order that the bytes followed in extended memory.) When the "end of time record" byte (decimal value 255) appears, lines **195 and 196** exit the loop by branching to line 215. Lines **197 and 198** branch to line 201 unless the Alpha register was empty. If the Alpha register is empty and the "end of time record" byte hasn't appeared, there are more bytes in this time record to recall from extended memory; line **199** recalls up to 24 more bytes and line **200** branches to line 192.

```
201♦LBL 01
202 X=Y?
203 CLX
204 ST+ 06
205 ST+ 07
206 .1
207 %
208 +
209 RCL IND 05
210 FRC
211 +
212 STO IND 05
213 ISG 05
214 GTO 08
```

Lines **201 through 214** store each day's byte value in main memory. Lines **202 and 203** change any "no hour" bytes (decimal value 254) to a decimal value of zero. Lines **204 and 205** add the byte value to R_{06} (total hours for this job) and R_{07} (total hours for all jobs). For a byte value of *nnn*, lines **206, 207, and 208** return *nnn.nnn* to the X-register. Lines **209 and 210** recall *.ppp*, where *ppp* is the accumulated byte value for all jobs on this day. Lines **211 and 212** place *nnn.qqq* in the register for this day, where $qqq = nnn + ppp$. Lines **213 and 214** increment the loop-control number and, unless the summary period has been completed, branch to line 192.

Displaying (Printing) Each Job's Summary. After each job is recalled from extended memory to main memory, the following routines display (print) that job's summary before recalling the next job.

```
215♦LBL 02
216 8.007
217 RCL 05
218 INT
219 9
220 −
221 +
222 STO 05
223 RCL 02
224 LASTX
225 CHS
226 DATE+
```

Lines **215 through 226** prepare to display (print) the daily hours for this job. Line **216** enters a loop-control number representing a single execution of the next routine: accessing only R_{08} (for only *most recent date*). Lines **217 through 220** calculate how many registers above R_{08} (for days before *most recent date*) received data for this job. (The job may not have had data for the beginning of the summary period.) The result *m* is used twice. First, lines **221 and 222** place a loop-control number $(8 + m).007$ in R_{05}, where R_{8+m} is the largest-addressed register that received a byte value for this job. (Line 217 recalls the final value of the loop-control number from the previous routine; the register that the previous routine accessed last will accessed first by the next routine.) Second, lines **223 through 226** subtract *m* days from *most recent date*; the result is the earliest date to be summarized for this job.

```
227♦LBL 09
228 RCL IND 05
229 INT
230 XEQ 00
231 DSE 05
232 GTO 09
233 RCL 06
234 XEQ 01
235 RCLPT
236 INT
237 1
238 +
239 SF 25
240 SEEKPT
241 FS?C 25
242 GTO 07
```

Lines **227 through 242** display (print) the summary for this job. The program executes a loop from line 228 through line 232 once for each day. Lines **228 and 229** recall the byte value of the hours for this job on this day. Line **230** calls a subroutine on line 264 that displays (prints) the date and hours. Lines **231 and 232** decrement the loop-control number and, if there are more days, branch to line 227. Line **233** recalls the byte value of the total hours for this job, and line **234** calls a subroutine on line 278 that displays (prints) **TOTAL** and the hours. Lines **235 through 240** attempt to set the pointer in TRECS to the name of the next job. If there is another job in TRECS, lines **241 and 242** branch to line 165.

Displaying (Printing) the Final Summary. After summarizing all jobs, the program displays (prints) a final summary of total hours for each day and total hours for the entire summary period.

```
243♦LBL 10
244 RCL 04
245 .007
246 +
247 STO 05
248 ADV
249 ADV
250 "SUMMARY"
251 AVIEW
252 RCL 03
```

Lines **243 through 252** prepare to display (print)) the final summary. Lines **244 through 247** store *nnn*.007 in R_{05}, where R_{nnn} is the register containing the daily hours for *earliest date*. Lines **248 and 249** advance the printer twice. Lines **250 and 251** display (print) **SUMMARY**. Line **252** recalls *earliest date*.

```
253♦LBL 11
254 RCL IND 05
255 FRC
256 1 E3
257 *
258 XEQ 00
259 DSE 05
260 GTO 11
261 RCL 07
262 XEQ 01
263 GTO 03
```

Lines **253 through 263** display (print) the final summary. The program executes a loop from line 254 through line 260 once for each day. Lines **254 through 257** recall the byte value of the total hours on this day, and line **258** calls a subroutine on line 264 that displays (prints) the date and hours. Lines **259 and 260** decrement the loop-control number and, if there are more days to summarize, branch to line 253. Line **261** recalls the byte value of the total hours for the entire summary period, and line **262** calls a subroutine on line 278 that displays (prints) **TOTAL** and the hours.

Subroutines for Displaying (Printing). These three routines display (print) the date, TOTAL, and the hours respectively. The program calls the first and second as subroutines, both of which transfer execution to the third, which finally returns execution to the next line after the subroutine call.

```
264♦LBL 00
265 X<>Y
266 1
267 RCL Y
268 " "
269 ADATE
270 "├ "
271 DOW
272 X=Y?
273 ADV
274 RDN
275 DATE+
276 X<>Y
277 GTO 02

278♦LBL 01
279 "TOTAL "

280♦LBL 02
281 10
282 /
283 LASTX
284 X>Y?
285 "├ "
286 X↑2
287 X>Y?
288 "├ "
289 ARCL Y
290 AVIEW
291 RCL Z
292 RTN
```

Lines **264 through 277** place the date in the Alpha register. This subroutine is called by lines 230 and 258; it expects the appropriate date in the Y-register and the byte value for the daily hours in the X-register. Lines **265, 266, and 267** place the date in X- and Z-registers, 1 in the Y-register, and the byte value in the T-register. Lines **268, 269, and 270** place the date in the Alpha register. Lines **271, 272, and 273** advance the printer if the date is a Monday. Line **274** rolls 1 into the X-register, the date into the Y-register, and the byte value into the Z-register. Line **275** calculates the succeeding date (preparing for the loop's next execution). Line **276** returns the byte value to the X-register (where it was before this routine). Line **277** branches to line 280.

Lines **278 and 279** place TOTAL in the Alpha register. This subroutine is called by lines 234 and 262.

Lines **280 through 292** append the hours to the Alpha register and display (print) the combined contents. Lines **281 and 282** convert the byte value to hours and tenths of hours. Lines **283 through 288** ensure that the hours will be printed right-justified. Line **289** appends the hours to the Alpha register and line **290** displays (prints) the combined contents. Line **291** recalls the next date to the X-register (preparing for the loop's next execution). Line **292** returns execution to the line 231, 235, 259, or 263.

Clearing Days or Jobs.

```
293♦LBL 03
294 CF 21
295 CLA
296 14
297 PASN
298 15
299 PASN
300 25
301 PASN
302 SF 27

303♦LBL 12
304 "CLEAR D, J? E"
305 PROMPT
306 TONE 5
307 GTO 12
```

Lines **293 through 302** prepare to display the clearing menu. Line **294** clears the Printer Enable flag. Lines **295 through 301** cancel any assignments to the [D], [E], and [J] keys. Line **302** activates the User keyboard.

Lines **303 through 307** display the clearing menu. Lines **304 and 305** prompt you to press [D] (*clear days*), [J] (*clear a job*), or [E] *(exit).* (When you press one of these keys, execution starts at the corresponding label.) If you mistakenly press [R/S], line **306** sounds a tone and line **307** branches to line 303.

Clearing a Job.

```
308♦LBL J
309 CF 23
310 "NAME OF JOB?"
311 AON
312 PROMPT
313 AOFF
314 FC?23
315 GTO 12
316 CLX
```

Lines **308 through 316** prepare to clear a job. Lines **309 through 312** prompt you for the name of the job that you want to clear. Lines **311 and 313** automatically activate and deactivate the Alpha keyboard. If you press [R/S] without entering a name, lines **314 and 315** branch to line 303. Line **316** provides the pointer value for the beginning of TREC as an initial value for the next routine.

```
317♦LBL 13
318 SEEKPT
319 POSFL
320 X<0?
321 GTO 00
322 2
323 MOD
324 X=0?
325 GTO 01
326 RCLPT
327 INT
328 1
329 +
330 GTO 13
```

Lines **317 through 330** locate the job to be cleared. Lines **318 and 319** search TRECS for the job you named in the Alpha register, starting the search at the pointer value in the X-register. If the name doesn't exist in TRECS, lines **320 and 321** branch to an error routine on line 331. If the pointer value returned by line 319 is valid for a name, lines **322 through 325** branch to line 337. If the pointer value is not valid (that is, if bytes in a time record spell the name by coincidence), lines **326 through 330** calculate the pointer value for the next record and branch to line 317.

```
331♦LBL 00
332 TONE 5
333 "NO SUCH NAME"
334 AVIEW
335 PSE
336 GTO 12
```

Lines **331 through 336** are an error routine. Line **332** sounds a tone; lines **333, 334, and 335** display an error message; and line **336** branches to line 303.

```
337♦LBL 01
338 DELREC
339 DELREC
340 GTO 12
```

Lines **337 through 340** clear the job from TRECS. Line **338** clears the job's name and line **339** clears the job's time record. Line **340** branches to line 303.

Clearing Days.

```
341♦LBL D
342 SF 00
343 XEQ 20
344 1.001
345 RCL 01
346 X<Y?
347 X<>Y
348 CLA
349 255
350 XTOA
```

Lines **341 through 350** prepare to clear days. Lines **342 and 343** call a subroutine on line 08 that determines the earliest and most recent dates to clear. (The primary purpose of the subroutine on line 08 is to determine the summary period; setting flag 00 modifies the subroutine's operation to determine the clearing period.) Lines **344 through 347** return 1.*ccc*, where *ccc* is the character-pointer value for the most recent date to clear. Line **344** enters the minimum value—at least one day's byte must be left in each time record—and line **345** recalls the value that corresponds to the most recent date you chose to clear. Lines 346 and 347 place the greater of these values in the X-register. Lines **348, 349 and 350** place 255 in the X-register and the "end of time record" byte in the Alpha register.

```
351♦LBL 14
352 DELCHR
353 INSCHR
354 X<>Y
355 2
356 +
357 SF 25
358 SEEKPT
359 FC?C 25
360 GTO 12
361 X<>Y
362 GTO 14
```

Lines **351 through 362** clear days from each job's time record. The program executes this loop once for each time record that contains data for the clearing period. Line **352** clears all bytes in the summary period, and line **353** inserts a new "end of time record" byte. Lines **354 through 358** try to set the pointer to the byte for the most recent date to clear in the next job's time record. If there isn't a byte at that location, lines **359 and 360** branch to line 303. (This will occur when there are no further records or when this and all following jobs have no data for the clearing period.) Line **361** returns 255 to the X-register and line **362** branches to line 351.

Exiting Σ.

```
363♦LBL E
364 RCL 00
365 STOFLAG
366 CLX
367 END
```

Lines **363 through 367** conclude Σ. Lines **364 and 365** return the flags to their previous status. Line **366** clears the flag-status data from the X-register.

Appendices

Appendix A

Error and Status Messages

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

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 Time Extended Memory Any other function using numeric data	Nonnumeric data was used for a function needing numeric data: the X-register (or Y- or Z-register, if relevant) contains Alpha data.
CHKSUM ERR	GETP GETSUB	Part of the program file has been lost.
DATA ERROR	Mathematical SDEV	Illegal math operation with the given operands (division by zero, square root of a negative number).
	Time	Invalid number in the X-register.
	y^x	$x \leqslant 0$ and $y = 0$, or x is noninteger and $y < 0$.
	T+X	$x < -9999.595999$ or $x > 9999.595999$, or ***MM*** or ***SS*** > 59.
	TONE	$x \geqslant 10$ or $x < 0$.
	MEAN	$n = 0$.

Display	Functions	Meaning
DATA ERROR (continued)	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	$\lvert n\rvert \geq 10$.
	GETKEYX	$x \geqslant 100$.
	AROT POSA XTOA X<>F	$x > 255$.
	PSIZE CLRGX SEEKPT SEEKPTA SWPT	$x > 999$.
	CRFLAS CRFLD	$x = 0$. (Attempted to create a file zero registers long.)
	EMDIRX CLALMX RCLALM RSZFL	$x = 0$ or $x > 999$.
	STOFLAG	x (or y, if x is a range of flags in the form *bb.ee*) was not obtained by executing RCLFLAG.
	X=NN? X≠NN? X<NN? X<=NN? X>NN? X>=NN?	Y-register contains Alpha data other than X, Y, Z, T, or L, or $y > 999$.
DATA ERROR X	DDAYS	x is an invalid or negative date, or the year portion uses more than four digits.
	XYZALM	$x \geqslant 24$ or is not a valid ***HH.MMSS*** value.
DATA ERROR Y	DATE+ DDAYS XYZALM	y is an invalid or negative date, or the year portion uses more than four digits.

Display	Functions	Meaning
DATA ERROR Z	XYZALM	The *z*-value ⩾10,000 hours or is not a valid ***HHHH.MMSS*** value.
DUP FL (*duplicate file*)	Extended Memory	A file of the same name already exists in extended memory. Text, data, and program files cannot use the same names. (The named file becomes the current file.)
END OF FL (*end of file*)	Extended Memory	A nonexistent pointer address was used during an attempt to position the pointer in a file, or read, write, or delete within a file. (With SAVER and SEEKPTA the named file becomes the current file, although the file and pointers are not changed.)
	APPCHR APPREC INSCHR INSREC	Not enough room to add the character or record.
	GETAS SAVEAS	File transfer not completed because the end of the destination file was encountered before the end of the source file. Part of the file was transferred.
END OF REC (*end of record*)	SEEKPT SEEKPTA	The end of the record was encountered during an attempt to position the character pointer.
ERROR=D*nn** (*delta split error in* R_{nn})	SW	The number stored in R_{nn} or R_{nn-1} is not in ***HH.MMSS*** format; or the split stored in R_{nn} is smaller than the split stored in R_{nn-1}.
ERROR=R*nn** (*split error in* R_{nn})	SW	The integer portion in R_{nn} exceeds 99.
FL NOT FOUND (*file not found*)	Extended Memory	The file specified (which might be the current file) does not exist in extended memory. (After purging a file there is no current file.)
FL SIZE ERR (*file size error*)	RESZFL	The new size specified (*x*) is smaller than the current size, and would eliminate some data (see page 213).
FL TYPE ERR (*file type error*)	Extended Memory	The file specified (which might be the current file) is of the wrong type for the function attempted.

* This error message is for information only, and does not have the effect of a true error condition. Refer to section 17.

Display	Functions	Meaning
KEYCODE ERR (*keycode error*)	PASN	The key specified (by keycode) is not assignable.
MEMORY LOST		Continuous Memory has been cleared and reset.
NAME ERR (*file name error*)	Extended Memory	No file name specified (the Alpha register is empty), or the Alpha register contains seven bytes of decimal value 255 (illegal).
	PCLPS SAVEP	The named program does not exist in main memory.
NO	Flags Conditionals	*Status message.* The result of a flag test or conditional test is false.
NO DRIVE (*no drive device present*)	GETAS SAVEAS	No HP-IL module is plugged in, or no mass storage device is on the interface loop.
NONEXISTENT	Storage Recall Extended Memory Conditionals CLRGX SWPT	One or more registers specified do not exist in data storage.
	ASN GTO XEQ Alarms	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 Alarms	The function called does not exist. For alarms, the function called must be a programmable catalog-2 function. If a catalog-2 function is called, its source device must be attached to the HP-41.
	STOFLAG	One or more flags specified are outside the range 0 to 43.
NO ROOM	Extended Memory	There is not enough room in extended memory for the program or file specified.
	GETP GETSUB	There is not enough room in main memory for the specified program.
	PSIZE	(When executed as a program instruction.) Not enough room remaining in main memory.

Display	Functions	Meaning
NO ROOM (continued)	RESZFL	Not enough room in extended memory to increase the file size.
	XYZALM	Not enough room in uncommitted memory to set the alarm; or, the maximum number (253) of registers for alarm storage would be exceeded.
	Text Editor*	No room left to add more characters or records.
NO SUCH ALM (*no such alarm*)	Alarms	The alarm specified does not exist.
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 = ±9.999999999 99.
	DATE+ T+X	The resulting date is outside the allowed calendar range.
PACKING **TRY AGAIN**	ASN PASN XEQ GTO . . GETP GETSUB 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 PSIZE	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).

* This error message is for information only, and does not have the effect of a true error condition. Refer to section 14.

Display	Functions	Meaning
REC TOO LONG	Text Editor APPCHR INSCHR	Attempted to exceed the maximum record length (254). (Deactivates the Text Editor if it was active.)
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.

Appendix B

More About Past-Due Alarms

Contents

When an alarm becomes past-due for one of the reasons described in section 16 under "Past-Due Alarms," it is maintained in memory until it is activated or until you delete it. This operation reminds you of an alarm that has not been allowed to serve its intended purpose. If you allow several past-due alarms to accumulate in memory, sequences of automatic past-due alarm activations can occur. (If any bypassed past-due alarms are in memory, the order in which past-due alarms activate can become complex.) If you plan to use past-due alarms or simultaneous alarms in your applications, the information in this appendix will be helpful. Simultaneous alarms activate in the same sequence as past-due alarms.

If past-due alarms are present, they will automatically begin to activate whenever you turn off the computer or press [ON]. This operation is to remind you that one or more past-due alarms exist. For the same reason, if an alarm comes due while any *bypassed* past-due alarms exist, all of the bypassed past-due alarms will activate ahead of the alarm that came due.* This appendix describes the rules governing the activation sequences in these two cases. (Refer to section 16 for a description and classification of past-due alarms.)

* A bypassed past-due alarm, as defined in section 16, is an alarm that never went off (never activated) because it was set to the past or because it was bypassed due to a time-change function ([SETIME], [SETDATE], [T+X], or [CORRECT]).

Conditions That Cause Execution of Past-Due Alarms

Turning off the computer or displaying the clock with [ON] initiates the off/clock condition. If any past-due alarms exist when this condition occurs, the computer attempts to activate all of them, beginning with the earliest alarm. If any past-due control or conditional alarm is encountered, the computer turns off momentarily—which aborts the off/clock condition—then turns back on in the alarm condition described under the heading, "Alarm Condition." The control/conditional alarm is then executed in the alarm condition.

Off/Clock Condition

In the off/clock condition:

- *As long as no past-due control/conditional alarms are encountered,* any past-due message alarms will go off in chronological order, beginning with the earliest alarm time. Each alarm will finish its activation cycle before the next alarm activates. Such alarms will not interrupt each other.
- If the [ON] key is pressed when a past-due message alarm goes off, the alarm halts without being acknowledged and the computer turns off or displays the clock.
- If no control/conditional alarms are past-due, and [ON] is not pressed during activation, the computer turns off or displays the clock after activating all the past-due alarms.

Alarm Condition

The alarm condition is initiated when:

- A future alarm comes due.
- A past-due control or conditional alarm (from the off/clock condition) starts a program or executes a function.

In the alarm condition, the computer activates only the bypassed past-due alarms in memory, beginning with the earliest alarm and proceding in chronological order of the alarm times. Other past-due alarms are ignored and remain in memory. When a future alarm comes due while there are bypassed past-due alarms, the computer switches to this alarm condition and the future alarm becomes, in essence, a bypassed past-due alarm. It will be activated in its turn, after all of the earlier bypassed past-due alarms are activated.

Past-Due Alarm Responses in the Alarm Condition

Listed below are terms used in the rest of this appendix to describe modes of the computer that affect alarm response.

Past-Due Alarm Responses

Alarm Type	Computer Mode			
	Off	Clock	Keyboard	Running
Conditional (↑)	Runs program.	Runs program.	Tone series and flashing display.	Sounds two tones and becomes past-due.
Control (↑↑)	Runs program.			Runs program as a subroutine of current program.
Message	Tone series and flashing display.			

- Off: the computer is turned off.
- Clock: the clock is displayed.
- Keyboard: the computer is turned on but is not displaying the clock or running a program.
- Running: A program is running.

The above table summarizes the computer's response when an alarm comes due in each of the modes described above.

When the alarm condition occurs, the computer's response to the various bypassed past-due alarms is determined by the current mode of the computer and by the alarm type.

Mode Changes

The program or function specified by an activating control or conditional alarm can change the computer operating mode:

- If any control or conditional alarm starts a program, the computer immediately switches to Running mode. (The mode change occurs before the first program instruction is executed.)
- A function executed by a control/conditional alarm can also change the mode. For example, if the clock is displayed when an alarm that executes the printer function PRX (*print X*) activates, the computer will change from Clock mode to Keyboard mode. Similarly, if the computer is executing a program when an alarm that executes the CLOCK function activates, the computer will change from Running mode to Clock mode.

Interruption of a Past-Due Alarm by Another Past-Due Alarm

- A program started by any past-due control or conditional alarm will be temporarily suspended by any subsequent bypassed past-due alarms before the first program instruction is executed.
- A message alarm or a function started by a past-due control/conditional alarm cannot be interrupted by a bypassed past-due alarm.

Alarms and Subroutine Levels

Any program alarm that interrupts a previous program alarm will operate as a subroutine. If there are several past-due control alarms that execute programs (which is unlikely, since any past-due control alarm would be a bypassed past-due alarm), then several subroutine levels will be used.

Example of a Past-Due Alarm Sequence

Suppose that the computer is turned off, the current time is 9:59 a.m., and the following four alarms are set:

Alpha Register	Time	Status
MESSAGE1	4:00 a.m.	Past-Due
↑ABC	5:00 a.m.	Bypassed Past-Due
↑↑XYZ	6:00 a.m.	Bypassed Past-Due
MESSAGE2	10:00 a.m.	Set to a Future Time

Note: The situation given in this example is unlikely, since bypassed past-due alarms do not occur in most applications. However, this mix of alarms helps to illustrate additional aspects of alarm response.

When the current time reaches 10:00 a.m. the MESSAGE2 alarm causes the Alarm condition to occur. Because there are bypassed past-due alarms, the following sequence occurs:

1. Alarm ↑ABC (the oldest *bypassed* past-due alarm) starts program ABC. (This is the first alarm to activate. The earlier MESSAGE1 alarm is not a bypassed past-due alarm and therefore will not be activated.)
2. Alarm ↑↑XYZ immediately suspends program ABC and starts program XYZ as a subroutine.

3. Alarm MESSAGE2 (which is now a bypassed past-due alarm) immediately suspends program XYZ, begins flashing **MESSAGE2** in the display, and, if not acknowledged from the keyboard, begins sounding a series of tones.
4. After alarm MESSAGE2 is acknowledged (or finishes its cycle), program XYZ is executed. Control then returns to program ABC (assuming that program XYZ did not stop or turn off the computer or use too many subroutine levels).
5. Program ABC is executed.

If program XYZ turns off the computer (by executing [OFF]), program ABC will not be resumed. Since alarm ↑ABC has already activated, it no longer exists in memory.

If alarm XYZ had been a conditional alarm it would have activated only by sounding a pair of tones and becoming a regular past-due alarm (since alarm ↑ABC would have switched the computer to Running mode). Refer back to the table of "Past-Due Alarm Responses" and to "Mode Changes". As a general guideline, whenever a past-due control or conditional alarm activates and starts a program, *any* subsequent past-due conditional alarm(s) will activate only by sounding the pair of tones and becoming past due alarm(s).

If the computer had been in Keyboard mode rather than off, alarm ↑ABC would have activated like a message alarm, displaying **↑ABC**. Program XYZ would then have started (but not as a subroutine), have been interrupted by alarm MESSAGE2, and finally have been executed.

Appendix C

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. However, if the Alpha string is rotated until an embedded null becomes a leading null, that null and any immediately following nulls will be lost.

---

* 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.

An embedded null in an Alpha string in the X-register signals the end of the string for which the function [POSA] (*position in Alpha*) will search to match in the Alpha register. (That is, the computer will match only that portion of the X-register string that is to the left of the first null.)

File Names. Any null embedded in a file name in the Alpha register is ignored by functions using that file name.

Appendix D

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.

The Time and Date on Program Listings

If a program is printed on an HP-IL printer (HP 82162A) using the HP 82160A HP-IL Module functions [PRP] or [LIST], the time and date appear in the display and are printed at the head of the program listing. (The time and date in the display are not in the X-register; press [←] to return to the X-register.)

Appendix E

Extended Memory Modules

Contents

Using the Modules

CAUTION

Always turn off the computer before inserting or removing any modules. Otherwise, the computer might be damaged or its operation might be disrupted.

The HP 82181A Extended Memory Module is identified by the legend X MEMORY on the module.

Legal Configurations

If you add only one extended memory module to your computer, the module can be installed in any of the four ports in the computer. Follow the steps for installation that are given below.

If you add a second extended memory module, the modules must be arranged in one of the following configurations. Don't install one above the other.

Legal Configurations for Two HP 82181A Extended Memory Modules

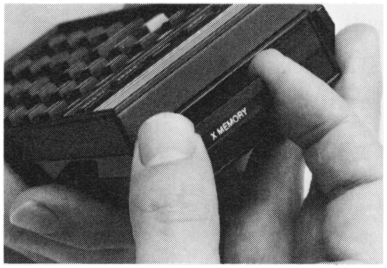
X MEMORY	X MEMORY

X MEMORY	X MEMORY

X MEMORY	
	X MEMORY

	X MEMORY
X MEMORY	

Installing the Modules



To insert an extended memory module:

1. **Turn off the computer!**
2. Hold the computer with the keyboard facing up.
3. Remove the cover from the port to be used.
4. Hold the module so that "X MEMORY" reads right side up.
5. Gently insert the module straight into the port.

You will feel the module snap into place when it's properly seated.

Removing the Modules

To remove an extended memory module:

1. **Turn off the computer!**
2. Use your fingernail to gently extend the extractor handle.
3. Grasp the handle and pull the module straight out of the computer.
4. Replace the cover on the open port.

The information in an extended memory module is lost when the module is removed from the computer. **You can lose all data in two extended memory modules even if you remove only one of the two modules.** To avoid this, observe the following rules when choosing which of two modules to remove.

- If one module was installed and used before the second module was installed, remove the second module rather than the first.
- If both modules were installed at the same time, remove the module in port 2 or port 4 rather than the module in port 1 or port 3. (The numbering of the ports is shown on the bottom of the computer.)

Before removing an extended memory module you can calculate which files will be lost. To save an important file you can purge earlier files until the following procedure shows that your important file will be saved:

1. Check the number of registers in each file using EMDIR, described on page 206.
2. For each file, add two extra registers for the header.
3. Add the total registers in the second file to the total registers in the first file to calculate the "address" of the end of the second file.
4. Add the total registers in each subsequent file to the previous total, calculating the "address" of the end of each file.

All files that end at an "address" exceeding 364 are lost when you remove one of two extended memory modules; all files that end at an "address" exceeding 126 are lost when you remove both memory modules.

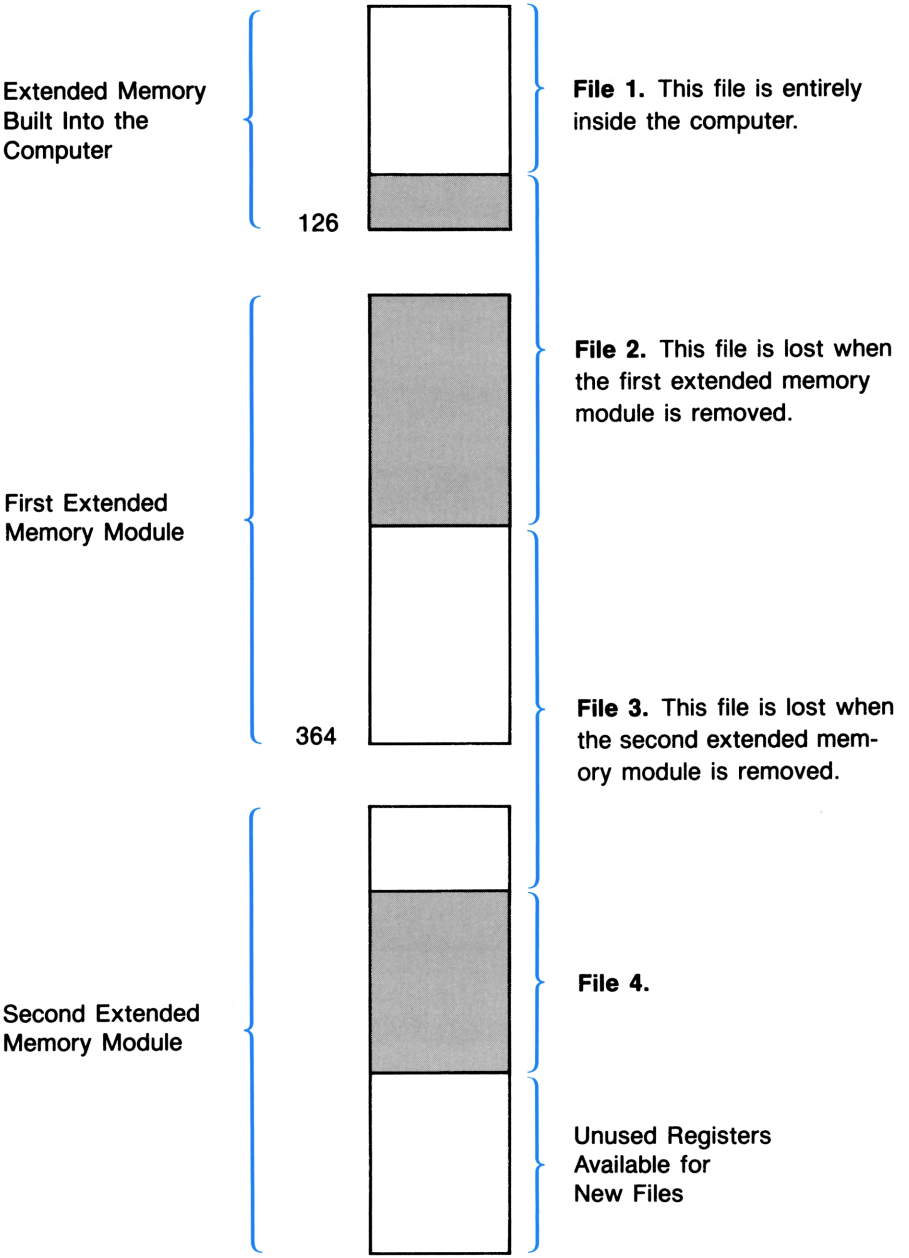
Map of Extended Memory

Shown below is a map of extended memory in a computer with two extended memory modules installed. The top block represents the 126 registers of extended memory built into the computer. File 1 is entirely inside the computer, so it is saved even if both extended memory modules are removed.

The middle block represents the 238 registers in the first extended memory module (the module used first or else the module in port 1 or 3). If you remove this module, files 2, 3, and 4 are lost.

The bottom block represents the 238 registers in the second extended memory module (the module used second or else the module in port 2 or 4). If you remove this module, files 3 and 4 are lost.

Map of Extended Memory



Appendix F

Time Specifications

Contents

The Accuracy Factor

The programmable, time adjustment functions allow you to correct the current time setting and to set and monitor the clock accuracy factor.

Like most timekeeping devices, the accuracy and precision of the HP-41 timer can be affected by variations in power supply, temperature, and manufacturing processes. While the effects of these variations are small, you might want to use the built-in accuracy factor to help compensate for the conditions affecting the time, *if these conditions will be fairly constant.* The accuracy factor (used with the CORRECT, RCLAF, and SETAF function) is meant to compensate over the *long term* for variations from the ideal, and not to compensate for anomolous conditions (such as exposure to freezing temperatures for several days). In the latter case, just reset the time (T+X) instead, if necessary.

The clock in the HP-41CX is basically the same as the clock in a quartz crystal watch. The accuracy of the HP-41CX clock, when adjusted with the accuracy factor, is similar to that of a quartz crystal watch. The unadjusted accuracy of the HP-41CX clock is not quite as good as that of a quartz crystal watch, but is better than that of a spring watch.

The accuracy factor is the time interval (in seconds) at which one pulse (of approximately 9.8×10^{-5} second duration) is added to or subtracted from the clock's 10240 Hz time base. The table at the right shows the accuracy factor limits and format.

An accuracy factor of −10.5 would cause one pulse to be subtracted every 10.5 seconds. An accuracy factor of 0.1 would cause one pulse to be added every 0.1 second.

The Accuracy Factor

Accuracy Factor, *n* (seconds)	Effect	
±99.9 ⋮ ± 0.1	Adds/subtracts one pulse every *n* seconds.	↓ Increasing frequency of correction.
0.0	Default	No correction.

You can determine the appropriate accuracy factor automatically with the CORRECT function or by calculation.

For information concerning timer precision and accuracy, see "Specifications for Time Precision and Accuracy".

Correcting the Time With the Accuracy Factor (CORRECT)

The CORRECT (*correct the time*) function sets the time that you specify and automatically adjusts the accuracy factor. When you place a time value (***HH.MMSShh***) in the X-register and execute CORRECT:

- The clock is set to the specified time in the same way that it is when you execute SETIME.
- The accuracy factor is automatically adjusted using an internal calculation based on drift* and the time span since SETIME, SETDATE, SETAF (*set accuracy factor*), or CORRECT was last executed. The timer then begins to alter automatically and continuously the clock time base according to the newly adjusted accuracy factor.

When you execute CORRECT manually, the precision of the timesetting operation will vary with your keystroke execution. Execution takes place when the key that executes CORRECT is released.† The time span between the most recent execution of SETIME, SETDATE, SETAF, or CORRECT and the subsequent execution of CORRECT must be long enough to render keystroke precision error insignificant. In most cases this time span should be a minimum of 30 hours. Greater increases in the time span between executions of CORRECT increase the probability of a more reliable accuracy factor.‡

---

* Drift is deviation from the correct time due to variations in power supply, temperature, and material variables. The value that the timer uses for drift is the difference between the current clock time and the new clock time (specified in the X-register) at the moment that you execute CORRECT.

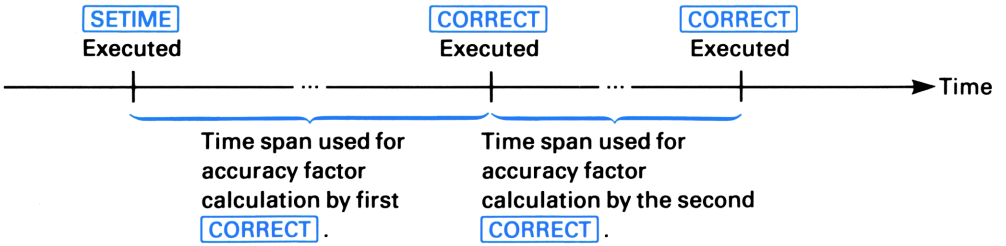
† Approximately ±0.1 second is the maximum keystroke precision for most users. You can reduce precision error by executing CORRECT as a function assigned to a key instead of by XEQ ALPHA **CORRECT** ALPHA. This is because the computer takes less time to internally locate and execute a function assigned to a single key.

‡ The longer you wait to execute CORRECT, the smaller the error due to keystroke variation becomes in proportion to any error resulting from a combination of all error factors. A practical time span for many applications is 1 week.

Note: The CORRECT function uses the calculated drift to determine the accuracy factor. For this reason, if you wish to use CORRECT to improve the long-term accuracy of the timer, you should not use T+X to remove time errors due to normal drift, because the alteration would not be detected by the CORRECT function. For this reason, using T+X to correct errors due to normal drift may result in a less reliable accuracy factor.

The accuracy factor adjustment performed by CORRECT depends in part upon the difference between the current time setting and the new time setting at the moment that you execute CORRECT. If the time has not been previously set using SETIME, executing CORRECT can result in an unfavorable accuracy factor. However, once the time has been initially set using SETIME, you can use CORRECT as often as is practicable.

The CORRECT Function



Remember that increasing the time span between execution of SETIME or CORRECT and execution of the next CORRECT will result in a more precise accuracy factor.

Recalling, Setting, and Clearing the Accuracy Factor

The RCLAF (*recall accuracy factor*) function recalls the current accuracy factor to the X-register. The stack is lifted in the same way as when you recall a number from a data storage register.

The SETAF (*set accuracy factor*) uses the value x put in the X-register to set the accuracy factor. The accuracy factor represents a time interval in seconds, as explained at the beginning of this section. The accuracy factor is either rounded to the nearest tenth of a second (***SS.t***) or set to zero, as follows:

- The accuracy factor will be set to 0.0 if $x = 0$ or $x \geqslant |99.95|$.
- The accuracy factor is set to 0.1 if $0 < x \leqslant 0.1$ ($x \neq 0$) and is set to -0.1 if $-0.1 \leqslant x < 0$.
- If the value x in the X-register is in the range $0.1 < x < 99.949$ or $-99.95 < x < -0.1$, the accuracy factor will be rounded to a $\pm$***SS.t*** format.

When you execute SETAF, the timer begins to automatically and continuously alter the clock time base according to the accuracy factor you specified.

To clear the accuracy factor, place 0 in the X-register and execute SETAF.

If you know that the HP-41 will have an interruption of power (such as removing the batteries), then you might want to recall the accuracy factor, write it down, and then use it to reset the accuracy factor when the power supply is normal again.

Accuracy Factor Calculation

The CORRECT function provides a convenient means to correct the timer's time base (through automatic calculation of the average accumulated error). However, if you want to establish an accuracy factor over a relatively short period of time (such as a 36-hour interval), any keystroke error that occurs when you execute CORRECT can have a more significant effect than when CORRECT is executed after longer intervals. By calculating the accuracy factor yourself, then entering it using SETAF, you can often implement a more effective accuracy factor over a shorter interval than you could by using CORRECT. Also, if you alter the drift by executing T+X, the accuracy factor that results from subsequently executing CORRECT is likely to be wrong. Thus, where drift has been altered by T+X, the best method of determining an effective accuracy factor would be by performing your own calculation.

You can calculate the accuracy factor using the following formula:

$$AF = \frac{1}{\frac{1}{IAF} - \frac{10240}{86400} ERR_{spd}}$$

Where:

IAF = initial accuracy factor
(If IAF is zero, substitute "0" for $1/IAF$.)

$ERRspd$ = the current error in seconds per day
(A "slow" clock has a negative error, and a "fast" clock has a positive error.)

10240 = clock internal time base pulse rate

86400 = the number of seconds in a day

After you calculate an accuracy factor, it should be rounded to one decimal place, then set using the SETAF function.

Specifications for Time Precision and Accuracy

Precision

Timesetting from the keyboard can be performed with a precision of about 0.1 second, but this can be less precise, depending upon human response time. The current *clock* setting can be adjusted with a precision of up to 0.01 seconds through use of the T+X function.

Accuracy

A crystal-stabilized time base provides accuracy control for the timer. As with any crystal-based timepiece, actual stability at any time is a function of operating temperature and voltage variations. If the computer is subjected to a consistent daily pattern of environmental conditions, the total inaccuracy can be made negligible through appropriate application and maintenance of the accuracy factor. The overall accuracy of the timer at 25°C is ±3.02 seconds per day (±35 ppm), not to change with age more than an additional ±1.30 seconds per day (±15 ppm).

Stopwatch

The precision for stopwatch time and difference between splits is ±0.01 seconds. For maximum accuracy, splits should be taken at intervals of no less than 0.08 seconds; otherwise, an error due to delays in internal processing time could result. Rapidly pressing the keys on the Stopwatch keyboard can cause temporary suppression of all or part of the stopwatch display, but does not affect stopwatch timekeeping ability.

Appendix G

Battery, Warranty, and Service Information

Contents

The Input/Output Ports

Keep the caps on the input/output ports whenever nothing is plugged into them.

CAUTION

Do not insert your fingers or any object other than an HP module or plug-in accessory into an input/output port. Doing so could interrupt Continuous Memory and possibly damage the computer.

Batteries and Power Use

The HP-41 is powered by four batteries. Depending on how it is used, the HP-41 can operate up to six months or more on a set of alkaline batteries. The batteries supplied with the computer are alkaline N cells, but a rechargeable battery pack (nickel cadmium cells) can also be used.

The total number of operating hours supplied by the batteries depends greatly on what kinds of operations you do and how much you use peripheral devices. Without using peripheral devices, a set of four fresh alkaline batteries will provide about 45 to 85 hours of *continuous* program running (the most power-consuming kind of computer use—refer to "Power Consumption," below). When only the display is on and no operations are being performed, much less power is consumed.

The actual lifetime of the batteries depends on how often you use the HP-41 and its peripherals, whether you use the HP-41 more for running programs or more for manual calculations, and which functions you use. Next to using peripheral devices, the most power-consuming operations are: running programs, displaying/running the stopwatch, using the catalogs, and displaying the clock. Catalogs 4, 5, and 6 draw as much power as a running program even when they are stopped (which is why they cause a faster time-out than usual).

If the computer remains turned off, a set of fresh batteries will preserve the contents of Continuous Memory for as long as the batteries would last outside of the computer—about 1½ years for alkaline batteries.

Power Consumption

The actual rate of power consumption depends upon how the HP-41 is being used at any given time. There are three basic power consumption modes:

- **Operating:** high current drain (5 to 20 mA). This corresponds to running a program, a catalog, or the Text Editor; displaying the stopwatch; or performing an operation (pressing a key).

- **Idle:** moderate current drain (0.5 to 2.0 mA). This mode corresponds to the display being on, including the display of the clock. If the clock shows only the time ([CLKT]), it consumes more power than if it shows the time and date ([CLKTD]). ([CLKT] updates the display more frequently.)
- **Off:** low current drain (0.01 to 0.05 mA). Exists when the computer is off. The timer's precision oscillator runs continuously to maintain the clock and, if running, the stopwatch.

While the computer is turned on, typical computer use is a mixture of idle time and operating time. Therefore, the actual lifetime of the batteries depends on how much time the computer spends in each of the three modes.

A freshly charged HP 82120A Rechargeable Battery Pack has a capacity of 65 mAH (milliampere-hours). A fresh set of alkaline batteries provides approximately 500 mAH.

Power Consumption by Peripheral Devices

When you use peripheral devices that draw power from the HP-41 batteries (such as the card reader or the optical wand), total battery life will be reduced considerably. If you use peripherals frequently, it is recommended that you power the HP-41 with an HP 82120A Rechargeable Battery Pack.

Effects of Clearing Memory, Power Interruptions, and Low Power

Clearing Memory. Resetting Continuous Memory ([←]/[ON]) does not affect:

- The clock and its format settings for [CLK12] versus [CLK24] and [CLKT] versus [CLKTD].
- The stopwatch.

Resetting Continous Memory does:

- Clear all of main and extended memories.
- Clear all alarms.
- Clear all User function assignments.
- Reset all flags to their initial, power-on settings. See the table "Summary of Flag Status" in section 19.
- Reset the allocation of registers in main memory to 100 registers for data storage.

Refer also to "Continuous Memory" in section 1 for information about clearing and resetting memory.

Temporary Power Interruption. A power interruption (including taking the batteries out) of sufficient duration can cause a *power reset*, which clears Continuous Memory *and* affects the time, date, and alarms. If affected, the time and date reset to 12:00 a.m. on January 1, 1900. Various other errors will be introduced into timer operation. For this reason it is recommended that after any power interruption you check the status of the current time and date. If the time and date are correct, then the integrity of the time data has been preserved. If the time and/or date are *not* correct, then do the following:

1. Reset the time and date (SETIME and SETDATE).
2. Reset the clock display formats, if they are not correct (CLK12 or CLK24 , CLKT or CLKTD).
3. Initialize the stopwatch (set to zero), and stop it if it is running (do 0 SETSW STOPSW).

Low Power. When battery power is too low to operate the clock display, executing CLOCK or ON will turn off the computer. (The clock will continue to keep time internally.) In most cases, this will not occur until the **BAT** annunciator is lit.

Low-Power Indication

The **BAT** (*battery*) annunciator appears in the display when the available battery power is running low. If a peripheral is in use, disconnecting it (after turning off both the HP-41 and the peripheral) will significantly extend battery life.

With alkaline batteries installed (and no peripheral attached):

- The computer can be used for about 2 to 7 hours of continuous program running after **BAT** first appears.*
- If the computer remains turned off, the contents of its Continous Memory will be preserved for about a month after **BAT** first appears.

Battery Replacement and Installation

The batteries supplied with the HP-41, as well as the alkaline batteries listed below for replacement, are *not* rechargeable.

WARNING

Do not attempt to recharge the batteries; do not store batteries near a source of high heat; do not dispose of batteries in fire. Doing so may cause the batteries to leak or explode.

* Note that this is the time available for *continuous operation*. If you are using the computer for manual calculations—a mixture of the idle and operating modes—the computer can be used for a much longer time after the **BAT** first appears.

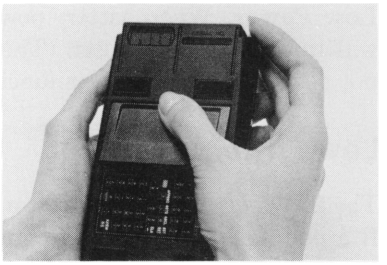
The following batteries are recommended for replacement in your HP-41:

Eveready E90*	Mallory MN9100	VARTA 7245
National AM5(s)	Panasonic AM5(s)	

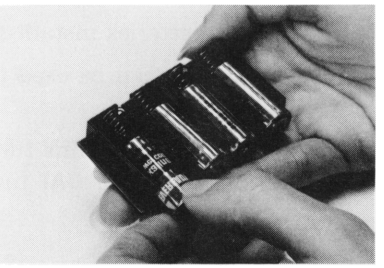
The contents of the computer's Continuous Memory are preserved for a short time while the batteries are out of the computer (provided that you turn off the computer before removing the batteries). This allows you ample time to replace the batteries without losing data or programs. If the batteries are left out of the computer for an extended period, the contents of Continuous Memory may be lost.

To install new batteries, use the following procedure:

1. Be sure the computer is off.
2. Holding the computer as shown, push up on the battery holder until it pops out.



3. Remove the batteries from the battery holder.

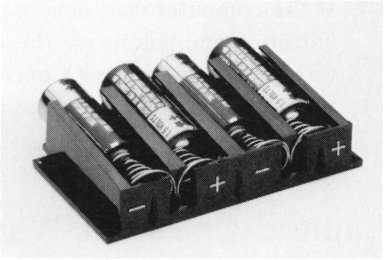


CAUTION

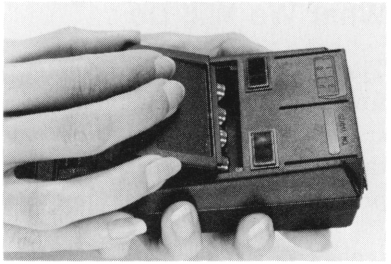
In the next step, replace all four batteries with fresh ones. If you leave an old battery inside, it may leak. Furthermore, be careful not to insert the batteries backwards. If you do so, the contents of Continuous Memory may be lost.

* Not available in the United Kingdom or Republic of Ireland.

4. Insert the new batteries by matching the position of their polarity marks to those on the battery holder. If any of the batteries are inserted backwards, the computer will not turn on.



5. Insert the battery pack into the computer such that the exposed ends of the batteries are pointing toward the input/output ports.
6. Push the upper edge of the battery pack into the HP-41 until it goes no further. Then snap the lower edge of the holder into place.



7. Turn the computer on. If for any reason memory has been cleared (that is, its contents have been lost), the display will show **MEMORY LOST**. Pressing any key will clear this message from the display.

Verifying Proper Operation

If it appears that the computer will not turn on or otherwise is not operating properly, review the following steps.

1. Be sure that all the batteries are inserted with the correct polarity and that the battery contacts are not dirty.
2. If the computer does not respond to keystrokes, try to reset it as follows: press and hold the [ON] and [ENTER↑] keys simultaneously, then release them. Turn the computer on, if necessary, and test for a response to keystrokes.
3. If there is no response, remove and reinsert the battery pack.

   If the computer still does not turn on, install fresh batteries.

   If this does not suffice, remove the battery pack and let the computer discharge overnight. When you reinstall the batteries and turn the computer on, if the display shows **MEMORY LOST**, then memory and the computer have been cleared and reset.

4. If the computer still does not respond to keystrokes, remove the battery pack and short the end battery terminals inside the HP-41 together. *Only momentary contact is required.* Replace the batteries. The contents of Continuous Memory will be lost, and you might need to press the [ON] key more than once to turn the computer back on.
5. If there is still no response, the computer requires service.

Limited One-Year Warranty

What We Will Do

The HP-41 is warranted by Hewlett-Packard against defects in material and workmanship for one year from the date of original purchase. If you sell your unit or give it as a gift, the warranty is automatically transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center.

What Is Not Covered

The batteries or damage caused by the batteries are not covered by this warranty. However, certain battery manufacturers may arrange for the repair of the computer if it is damaged by the batteries. Contact the battery manufacturer first if you computer has been damaged by the batteries.

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY. Some states, provinces, or countries do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES. Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state, province to province, or country to country.

Warranty for Consumer Transactions in the United Kingdom

This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

Obligation to Make Changes

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products once sold.

Warranty Information

If you have any questions concerning this warranty, please contact:

- In the United States:

Hewlett-Packard Company
Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.
Telephone: (503) 758-l0l0
Toll-Free Number: (800) 547-3400 (except in Oregon, Hawaii, and Alaska)

- In Europe:

Hewlett-Packard S.A.
150, route du Nant-d'Avril
P.O. Box
CH-1217 Meyrin 2
Geneva
Switzerland
Telephone: (022) 83 81 11
Note: Do *not* send computers to this address for repair.

- In other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Telephone: (415) 857-1501
Note: Do *not* send computers to this address for repair.

Service

Hewlett-Packard maintains service centers in most major countries throughout the world. You may have your unit repaired at a Hewlett-Packard service center any time it needs service, whether the unit is under warranty or not. There is a charge for repairs after the one-year warranty period.

Hewlett-Packard handheld computer products normally are repaired and reshipped within five (5) working days of receipt at any service center. This is an average time and could vary depending upon the time of year and work load at the service center. The total time you are without your unit will depend largely on the shipping time.

Obtaining Repair Service in the United States

The Hewlett-Packard United States Service Center for handheld and portable computer products is located in Corvallis, Oregon:

Hewlett-Packard Company
Service Department
P.O. Box 999
Corvallis, Oregon 97339, U.S.A.

or

1030 N.E. Circle Blvd.
Corvallis, Oregon 97330, U.S.A.

Telephone: (503) 757-2000

Obtaining Repair Service in Europe

Service centers are maintained at the following locations. For countries not listed, contact the dealer where you purchased your computer.

AUSTRIA
HEWLETT-PACKARD Ges.m.b.H.
Kleinrechner-Service
Wagramerstrasse-Lieblgasse 1
A-1220 Wien (Vienna)
Telephone: (0222) 23 65 11

BELGIUM
HEWLETT-PACKARD BELGIUM SA/NV
Woluwedal 100
B-1200 Brussels
Telephone: (02) 762 32 00

DENMARK
HEWLETT-PACKARD A/S
Datavej 52
DK-3460 Birkerød (Copenhagen)
Telephone: (02) 81 66 40

EASTERN EUROPE
Refer to the address listed under Austria.

FINLAND
HEWLETT-PACKARD OY
Revontulentie 7
SF-02100 Espoo 10 (Helsinki)
Telephone: (90) 455 02 11

FRANCE
HEWLETT-PACKARD FRANCE
Division Informatique Personnelle
S.A.V. Calculateurs de Poche
F-91947 Les Ulis Cedex
Telephone: (6) 907 78 25

GERMANY
HEWLETT-PACKARD GmbH
Kleinrechner-Service
Vertriebszentrale
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56
Telephone: (611) 50041

ITALY
HEWLETT-PACKARD ITALIANA S.P.A.
Casella postale 3645 (Milano)
Via G. Di Vittorio, 9
I-20063 Cernusco Sul Naviglio (Milan)
Telephone: (2) 90 36 91

NETHERLANDS
HEWLETT-PACKARD NEDERLAND B.V.
Van Heuven Goedhartlaan 121
NL-1181 KK Amstelveen (Amsterdam)
P.O. Box 667
Telephone: (020) 472021

SPAIN
HEWLETT-PACKARD ESPANOLA S.A.
Calle Jerez 3
E-Madrid 16
Telephone: (1) 458 2600

SWITZERLAND
HEWLETT-PACKARD (SCHWEIZ) AG
Kleinrechner-Service
Allmend 2
CH-8967 Widen
Telephone: (057) 31 21 11

NORWAY
HEWLETT-PACKARD NORGE A/S
P.O. Box 34
Oesterndalen 18
N-1345 Oesteraas (Oslo)
Telephone: (2) 17 11 80

SWEDEN
HEWLETT-PACKARD SVERIGE AB
Skalholtsgatan 9, Kista
Box 19
S-163 93 Spanga (Stockholm)
Telephone: (08) 750 2000

UNITED KINGDOM
HEWLETT-PACKARD Ltd
King Street Lane
GB-Winnersh, Wokingham
Berkshire RG11 5AR
Telephone: (0734) 784 774

International Service Information

Not all Hewlett-Packard service centers offer service for all models of HP computer products. However, if you bought your product from an authorized Hewlett-Packard dealer, you can be sure that service is available in the country where you bought it.

If you happen to be outside of the country where you bought your unit, you can contact the local Hewlett-Packard service center to see if service is available for it. If service is unavailable, please ship the unit to the address listed above under "Obtaining Repair Service in the United States." A list of service centers for other countries can be obtained by writing to that address.

All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Repair Charge

There is a standard repair charge for out-of-warranty repairs. The repair charges include all labor and materials. In the United States, the full charge is subject to the customer's local sales tax. In European countries, the full charge is subject to Value Added Tax (VAT) and similar taxes wherever applicable. All such taxes will appear as separate items on invoiced amounts.

Computer products damaged by accident or misuse are not covered by the fixed repair charges. In these situations, repair charges will be individually determined based on time and material.

Service Warranty

Any out-of-warranty repairs are warranted against defects in materials and workmanship for a period of 90 days from date of service.

Shipping Instructions

Should your unit require service, return it with the following items:

- A completed Service Card, including a description of the problem.
- A sales receipt or other proof of purchase date if the one-year warranty has not expired.

The product, the Service Card, a brief description of the problem, and (if required) the proof of purchase should be packaged in the original shipping case or other adequate protective packaging to prevent in-transit damage. Such damage is not covered by the one-year limited warranty; Hewlett-Packard suggests that you insure the shipment to the service center. The packaged unit should be shipped to the nearest Hewlett-Packard designated collection point or service center. Contact your dealer for assistance. (If you are not in the country where you originally purchased the unit, refer to International Service Information above.)

Whether the unit is under warranty or not, it is your responsibility to pay shipping charges for delivery to the Hewlett-Packard service center.

After warranty repairs are completed, the service center returns the unit with postage prepaid. On out-of-warranty repairs in the United States and some other countries, the unit is returned C.O.D. (covering shipping costs and the service charge).

Further Information

Service contracts are not available. Computer product circuitry and design are proprietary to Hewlett-Packard, and service manuals are not available to customers.

Should other problems or questions arise regarding repairs, please call your nearest Hewlett-Packard service center.

Technical Assistance

The keystroke procedures in this manual are supplied with the assumption that the user has a working knowledge of the concepts and terminology used. Hewlett-Packard's technical support is limited to explanation of operating procedures used in the manual and verification of answers given in the examples. Should you need further assistance, you may write to:

Hewlett-Packard Company
Portable Computer Division
Customer Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330

Dealer and Product Information

For dealer locations, product information, and prices, please call (800) 547-3400. In Oregon, Alaska, or Hawaii, call (503) 758-1010.

Temperature Specifications

- Operating: 0° to 45°C (32° to 113°F)
- Storage: 0° to 45°C (32° to 113°F).

*If the batteries are removed, or if clock accuracy is not a concern, then the storage temperature tolerances for the HP-41 are:**

−20° to 60°C (−4° to 140°F)

Potential for Radio and Television Interference (For U.S.A. Only)

The HP-41 generates and uses radio frequency energy and, if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If your HP-41 does cause interference to radio or television reception, you are encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna.
- Relocate the computer with respect to the receiver.
- Move the computer away from the receiver.

If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet prepared by the Federal Communications Commission helpful: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, D.C.20402, Stock No. 004-000-00345-4.

* If the clock's operation is affected by *temporary* exposure to extreme temperature, do *not* reset it using the accuracy factor; use [T+X] instead (that is, simply reset the time, and do not adjust the accuracy factor). As explained in appendix F, the accuracy factor compensates for the effects of fairly constant conditions, and should not be used to compensate for conditions that will fluctuate irregularly.

Appendix H

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 82181A Extended Memory Modules

An extended memory module provides an additional 238 storage registers (1,666 bytes) of extended memory to the HP-41 (the HP-41CX comes with 124 registers of extended memory). You can add one or two extended memory modules. For more information, refer to appendix E, "Extended Memory Modules."

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. With its timekeeping capabilities, the HP-41CX can conduct timed data collection by HP-IL devices and provide automatic output of results. 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," section 9) is a listing of all XROM functions/programs by device.* For the HP-41CX, catalog 2 shows only the names of the source devices (the "ROM headers") until you stop the catalog and press [ENTER↑]. This starts a listing of the individual functions and programs supplied by the last device whose name was displayed. To return to the listing of source devices, stop the catalog again and press [ENTER↑].

[SST] and [BST] work as for other catalogs. [SST] and [BST] will not cross from the header list to the function list (only [ENTER↑] does this). 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.

* Some internal functions, such as time functions and extended memory functions, are also part of catalog 2. Refer to "Time Functions" and "Extended Memory and Extended Functions" in appendix I.

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.*

- 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 → / ← plug module back in — **XROM** *nn,mm*

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 → / ← unplug again — **XEQ**^{T}*label*

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

* 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.

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.

Memory Space. An **XROM**^{T}*label* or *label* instruction (case A) requires two bytes of memory, while an **XEQ**^{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.* Even though they are *internal* (not plug-in) functions for the HP-41CX, all extended memory functions and extended functions use the XROM identification 25, and all time functions use the XROM identification 26. (None of the other internal functions have XROM numbers.)

* 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.

Appendix I

A Comparison With the HP-41C/CV

Contents

This appendix defines the differences between the HP-41CX and the HP-41C/CV so that you can quickly learn to use the HP-41CX if you are already familiar with the HP-41C/CV. If you have experience with the HP-41C or HP-41CV, then you already know much about the operation of the HP-41CX.

Programs written for the HP-41C/CV (including plug-in modules) are fully compatible with the HP-41CX. Programs written for the HP-41CX, however, are not necessarily compatible with the HP-41C/CV.

An Overview

The HP-41CX computer is based on the HP-41CV (which is like an HP-41C but with five times as much main memory space). It includes all the functions and memory space from the HP 82182A Time Module and the HP 82180A Extended Functions/Memory Module, plus additional alarm, stopwatch, extended memory, and other functions. The catalogs operate slightly differently compared with the HP-41C/CV, and the initial memory allocation is different: 100 registers for data storage. A point-by-point comparison is given below, including page references to explanations of features in this manual.

Cataloguing the New Functions

The HP-41CX functions are catalogued such that all new functions (functions not in the HP-41C/CV) are listed in catalog 2, the external-functions catalog, leaving catalog 3, the standard-functions catalog, unchanged from the HP-41C/CV. They are categorized in this way for consistency with previous products: the standard function set remains unchanged, and the time functions and extended functions are in catalog 2, as they are when they are supplied by the time module and extended functions/memory module. Therefore, all time and extended memory/extended functions in the HP-41CX use the XROM numbers (*external ROM numbers*; see appendix H) from the original modules for identification to the computer. The ROM identification number for all extended memory/extended functions is 25; the ROM identification for all time functions is 26. (Do not use a plug-in ROM module if it duplicates one of these identification numbers.)

The Owner's Manual

The owner's manual has also been completely rewritten for the HP-41CX. Many explanations in the previous literature (for the HP-41C/CV and for the two modules) have been changed, updated, and clarified. To this aim, some terminology used in this book is new, especially in the areas of programming, memory, and alarms. (See the table "Equivalent Terms" at the end of this section.) The printing conventions for shifted and nonkeyboard functions have also been changed. (See the inside of the front cover.)

Memory Configuration

The allocation of memory in the HP-41 computers is shown below.

Memory Configuration

Device	Main Memory			Total Extended Memory
	Total	Initial Configuration		
		Data Storage	Uncommitted*	
HP-41CX	319	100 (R_{00}–R_{99})	219	124
HP-41CV	319	273 (R_{00}–R_{272})	46	0
HP-41C	63	17 (R_{00}–R_{16})	46	0

* Memory for program instructions, alarms, and User function assignments are all drawn from the uncommitted registers. See section 12 for more information.

Catalog Operation

While an HP-41CX catalog is listing its contents, pressing any key besides [R/S] and [ON] will *speed up* the listing. (With the HP-41C/CV, this would *slow down* the listing.)

All the catalogs are summarized in section 9 under "The Catalogs."

Catalogs 1, 2, and 3

The display for catalog 1 now shows the number of bytes for each program (page 171). The display for catalog 2 is quite different, being broken up into function groups. (See "Time Functions" and "Extended Memory and Extended Functions" in appendix I.) Catalog 3 remains the same.

New Catalogs: 4, 5, and 6

The HP-41CX has three new catalogs. They all use power at the same rate as a running program, even when stopped, unlike catalogs 1, 2, and 3. They therefore will automatically terminate in 2 minutes (1 minute when the battery power is low) when they are stopped.

The new catalogs blink at the end of the listing when you try to use [SST], just as they blink at the beginning of the catalog when you try to use [BST]. (In the HP-41C/CV, [SST] at the end of the listing terminates the catalog function.)

Catalog 4: The Extended Memory Directory (Page 206). A listing of all files in extended memory can be accessed with either [CATALOG] 4 or [EMDIR]. [CATALOG] 4 is *not* programmable, however, while [EMDIR] is. The [EMDIR] function is essentially the same function as in the extended functions/memory module, *except* that now [R/S] will start and stop it, [SST] and [BST] will step through it, and a printer will only print it in Trace mode.

Catalog 5: The Alarm Catalog (Page 255). A listing of all alarms in memory can be accessed with either [CATALOG] 5 or [ALMCAT]. [CATALOG] 5 is *not* programmable, however, while [ALMCAT] is. The [ALMCAT] function is essentially the same function as in the time module, *except* that if there are no alarms in memory, the display shows **CAT EMPTY**.

Catalog 6: User Key Assignments (Page 168). A listing of all User key assignments for functions and global labels is given in order of keycode. Pressing [■] [C] will cancel a particular key assignment.

Time Functions

Part V ("Time Functions in Detail") contains all of the time functions in the HP-41CX: "Clock and Date Functions" (15), "Alarm Functions" (16), and "Stopwatch Operation" (17). In addition to the set of time functions included in the time module, there are more alarm operations, especially alarm-clearing procedures, and one extra stopwatch function:

- A repeating message alarm can be cleared from memory by pressing [C] to acknowledge it while the alarm is going off. (Page 255.)
- There are programmable alarm-clearing functions [CLALMA] (*clear alarm by Alpha*), [CLALMX] (*clear alarm by X*), and [CLRALMS] (*clear all alarms*). (Page 258.)
- The programmable function [RCLALM] (*recall alarm*) will recall the parameters of an alarm to the stack and Alpha register. (Page 252.)
- The minimum repeat interval for an alarm is 1 second instead of 10 seconds. (Page 250.)
- The stopwatch can be activated and the stopwatch pointers set with the programmable function [SWPT] (*stopwatch and pointers*). (Page 274.)

Those functions taken from the time module are listed in catalog 2 under **—TIME 2x** (*time functions, revision 2x*). The new time functions are in catalog 2 under **—CX TIME** (*HP-41CX time functions*).

Extended Memory and Extended Functions

The HP-41CX includes extended memory, extended memory functions, and extended functions, most of which are from the extended memory/functions module, and some of which are completely new. These capabilities are:

- 124 registers of *extended memory* for program, data, and text (ASCII) files.
- Functions for creating and operating on files in extended memory (*extended memory functions*).
- Functions that manipulate flags, data, and Alpha strings (*extended functions*).
- New conditional tests for branching (*extended functions*).
- Miscellaneous additional functions (*extended functions*).

The functions taken from this extended functions/memory module are catalogued in the HP-41CX in catalog 2 under the header **—EXT FCN 2x** (*extended functions, revision 2x*). The new, HP-41CX extended memory functions and extended functions are listed in catalog 2 under **—CX EXT FCN** (*HP-41CX extended functions*).

Extended Memory Functions

Section 13 ("Extended Memory") and section 14 ("The Text Editor") cover extended memory and the manipulation of files in extended memory: that is, the extended memory functions. The particular functions listed below are those new to the HP-41CX: the new extended memory functions.

The Text Editor is a major innovation in the HP-41CX. The function [ED] (*editor*) redefines the keyboard and display so that you can call up a text file and watch the contents of a record as you work on it, instead of having to manipulate text via single operations in the Alpha register. The keyboard for the text editor is reproduced on the backplate of the HP-41CX. This keyboard includes the Alpha character set. (The backplate on the HP-41C/CV shows only the Alpha keyboard.)

The other new extended memory functions are:

- ASROOM (*ASCII room*) and EMROOM (*extended memory room*), to return the amount of memory space left in an ASCII (text) file (page 222) or in extended memory (page 208).
- EMDIRX (*extended memory directory by X*), to recall the name and type of a certain file, and make it the current file (page 207).
- RESZFL (*resize file*), to change the size of a text or data file (page 213).

Extended Functions

The extended functions in the HP-41CX fall into the three general categories outlined below. Some of these functions were part of the extended functions/memory module. Those that were not are called the HP-41CX extended functions, as indicated below. They are listed in catalog 2.

Functions That Manipulate Flags, Data, and Alpha Strings. The HP-41CX includes functions from the extended functions/memory module to manipulate flags, data, and Alpha strings in and between registers. All functions for manipulating data are in section 12, "Main Memory." Functions to manipulate flags are in section 19, "Flags." Functions to manipulate Alpha data are in section 21, "Alpha and Interactive Operations."

Conditionals. The conditionals are new extended functions, allowing you to compare the value in the X-register with the value in any other data register. See section 20, "Branching."

Miscellaneous. The miscellaneous extended functions taken from the extended functions/memory module are:

- PASN (*programmable assign*) and CLKEYS (*clear key assignments*) in section 9, "The Keyboard and Display."
- SIZE? (*memory size?*) and PSIZE (*programmable size*) in section 12, "Main Memory."
- PCLPS (*programmable clear programs*) in section 18, "Programming Basics."
- GETKEY in section 21, "Alpha and Interactive Operations." This function halts program execution until a key is pressed, and that key's keycode can be used to branch to a particular subroutine.

The new, HP-41CX miscellaneous extended functions are:

- ΣREG? (*statistics registers?*) in section 11, "Numeric Functions."
- CLRGX (*clear registers by X*) in section 12, "Main Memory."
- GETKEYX (*get key by X*) in section 21, "Alpha and Interactive Operations."

New Terminology Used in This Manual

Many terms and names used in this manual are not the same as those used in previous literature for the HP-41 and its modules. If you are used to the previous terms, refer to the following list.

Equivalent Terms

HP-41CX	HP-41C/CV, Modules	Comments
Alpha execution	display execution	
Alpha name	display execution name	
bypassed past-due alarm	unactivated past-due alarm	
conditional alarm	noninterrupting control alarm	To emphasize the difference between conditional and control alarms.
control alarm	interrupting control alarm	
current file	working file	
flags: user (00–10) control (11–29)	 general-purpose user (00–10) special-purpose user (11–20)	User flags are strictly those defined by the user; control flags are defined by the HP-41. However, you can alter any of the user and control flags (but not the system flags, 30–55).
input cue	prompt	Avoids confusion with the PROMPT function.
keyboards: Alarm Catalog Alpha Normal Stopwatch User	modes	These particular conditions are characterized by redefined keyboards.
modes: Regular Split Delta Split Recall (splits) Storage (splits)	operations	These conditions define a mode of operation rather than a particular operation or function.
registers above R_{99}	extended storage registers	Avoids confusion with registers in extended memory.
text files	ASCII files	
uncommitted registers	program registers, program memory	This part of memory stores more than just programs.
GTO	■ GTO	All shifted functions are gold. See inside front cover.

Appendix J

Bar Code for Programs

Contents

SETALM (from section 16)
Program registers needed: 11

ROW 1 (1 : 3)

ROW 2 (3 : 7)

ROW 3 (7 : 9)

ROW 4 (10 : 12)

ROW 5 (13 : 22)

ROW 6 (23 : 27)

ALMREL (from section 16)
Program registers needed: 11

ROW 1 (1 : 3)

(continued)

ALMREL (continued)

ROW 2 (3 : 5)

ROW 3 (5 : 9)

ROW 4 (9 : 19)

ROW 5 (20 : 28)

ROW 6 (29 : 34)

SPLITS (from section 17)

Program registers needed: 14

ROW 1 (1 : 3)

ROW 2 (4 : 6)

ROW 3 (6 : 8)

ROW 4 (9 : 16)

ROW 5 (17 : 23)

ROW 6 (24 : 30)

ROW 7 (30 : 38)

ROW 8 (39 : 40)

TR (from section 22)
Program registers needed: 70

ROW 1 (1 : 6)

ROW 2 (6 : 12)

ROW 3 (12 : 15)

ROW 4 (15 : 20)

ROW 5 (20 : 22)

ROW 6 (23 : 31)

ROW 7 (32 : 38)

ROW 8 (39 : 43)

ROW 9 (43 : 48)

ROW 10 (49 : 54)

ROW 11 (54 : 57)

ROW 12 (58 : 62)

ROW 13 (62 : 68)

(continued)

TR (continued)

ROW 14 (68 : 78)

ROW 15 (78 : 85)

ROW 16 (86 : 93)

ROW 17 (94 : 98)

ROW 18 (98 : 103)

ROW 19 (103 : 111)

ROW 20 (111 : 117)

ROW 21 (118 : 124)

ROW 22 (125 : 133)

ROW 23 (134 : 137)

ROW 24 (137 : 144)

ROW 25 (145 : 151)

ROW 26 (152 : 155)

(continued)

TR (continued)

ROW 27 (155 : 163)

ROW 28 (163 : 169)

ROW 29 (169 : 177)

ROW 30 (177 : 183)

ROW 31 (184 : 185)

ROW 32 (185 : 194)

ROW 33 (194 : 202)

ROW 34 (203 : 213)

ROW 35 (214 : 222)

ROW 36 (222 : 227)

ROW 37 (228 : 230)

ROW 38 (230 : 233)

Σ (from section 22)
Program registers needed: 100

ROW 1 (1 : 7)

ROW 2 (7 : 12)

ROW 3 (13 : 18)

ROW 4 (18 : 25)

ROW 5 (25 : 34)

ROW 6 (34 : 37)

ROW 7 (37 : 44)

ROW 8 (44 : 51)

ROW 9 (51 : 56)

ROW 10 (56 : 64)

ROW 11 (65 : 71)

ROW 12 (72 : 79)

ROW 13 (80 : 88)

(continued)

Σ (continued)

ROW 14 (88 : 90)

ROW 15 (90 : 99)

ROW 16 (99 : 109)

ROW 17 (109 : 117)

ROW 18 (118 : 126)

ROW 19 (127 : 130)

ROW 20 (130 : 135)

ROW 21 (135 : 140)

ROW 22 (140 : 145)

ROW 23 (145 : 147)

ROW 24 (147 : 153)

ROW 25 (153 : 162)

ROW 26 (163 : 173)

(continued)

Σ (continued)

ROW 27 (174 : 181)

ROW 28 (181 : 190)

ROW 29 (190 : 197)

ROW 30 (198 : 205)

ROW 31 (206 : 214)

ROW 32 (214 : 222)

ROW 33 (223 : 231)

ROW 34 (231 : 239)

ROW 35 (239 : 245)

ROW 36 (246 : 251)

ROW 37 (252 : 259)

ROW 38 (259 : 267)

ROW 39 (267 : 274)

ROW 40 (275 : 279)

(continued)

Σ (continued)

ROW 41 (279 : 288)

ROW 42 (288 : 296)

ROW 43 (296 : 302)

ROW 44 (303 : 304)

ROW 45 (304 : 310)

ROW 46 (310 : 312)

ROW 47 (313 : 321)

ROW 48 (321 : 330)

ROW 49 (331 : 333)

ROW 50 (333 : 340)

ROW 51 (340 : 344)

ROW 52 (345 : 353)

ROW 53 (353 : 361)

ROW 54 (362 : 367)

Function Tables

Function Tables

Contents

Introduction

These ten 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"; the table for extended memory functions includes special entries described with that table.

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 [f] 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 [f]; 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.

Page. These are references to this volume. For references to volume 1, "Basic HP-41 Operation," see the Function Index inside the back cover.

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		155
AOFF		Deactivates Alpha keyboard.		E	48	1	159
AON		Activates Alpha keyboard.		E	48	1	159
ASN	ASN	Assigns specified function or global label to specified key on User keyboard.		E			166
CAT *n*	CATALOG *n*	Executes catalog *n*, $1 \leqslant n \leqslant 6$.					
		Catalogs 1, 2, 3, 6.		N			170
		Catalog 4 (EMDIR).		E		(2)	206
		Catalog 5 (ALMCAT).		E	31	(2)	255
CF *nn*	CF *nn*	Clears flag *nn*, $00 \leqslant nn \leqslant 29$.	I	E	*nn*	2	288
CLK12		Selects 12-hour clock display.		E		2	239
CLK24		Selects 24-hour clock display.		E		2	239
CLKEYS		Clears all assignments on User keyboard.		E		2	168
CLKT		Selects time-only clock display.		E		2	239
CLKTD		Selects time-and-date clock display.		E		2	239
DEG		Selects decimal Degrees angular mode.		E	42-43	1	186
DMY		Selects day-month-year date format.		E	31	2	242
ENG *n*	ENG *n*	Selects engineering display format with $n + 1$ digits.	I	E	36-41	2	161

System/Format Functions (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
FIX *n*	FIX *n*	Selects fixed-point display format with *n* decimal places.	I	E	36-41	2	160
GRAD		Selects Grads angular mode.		E	42-43	1	186
MDY		Selects month-day-year date format.		E	31	2	242
	ON	Turns computer on/off.			11-26, 45-55		155
ON		Selects continuous on (disables time-out).			44		155
PASN		Assigns function or label **specified in Alpha register** to User keyboard by key code **specified in X-register**.		E		2	166
	PRGM	Enters/exits Program mode.		N			155
PSIZE		Allocates *n* main memory registers for data storage, ***n*** **specified in X-register**.		E		2	199
RAD		Selects Radians angular mode.		E	42-43	1	186
RCLFLAG		Recalls status of flags 00 through 43.		↑, E		2	296
SCI *n*	SCI *n*	Selects scientific display format with *n* decimal places.	I	E	36-41	2	161
SF *nn*	SF *nn*	Sets flag *nn*, $00 \leqslant nn \leqslant 29$.	I	E	*nn*	2	288
ΣREG *nn*		Assigns statistics registers to R_{nn} through R_{nn+5}.	I	E		2	190
ΣREG?		Returns address of first currently defined statistics register.		↑, E		2	190
SIZE *nnn*		Allocates *nnn* main memory registers for data storage.		E			199

System/Format Functions (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
SIZE?		Returns number of main memory registers currently allocated for data storage.		↑, E		2	199
STOFLAG		Restores status of flags 00 through 43 using **flag-status data in X-register**, or:		E	00-43	2	296
		Restores status of flags *bb* through *ee* **specified by *bb.ee* in X-register** using **flag-status data in Y-register**.		E	*bb-ee*	2	296
	USER	Activates/deactivates User keyboard.		N	27		155

Clearing Functions

To interpret this table, refer to "Explanation of Table Entries" on page 414.

Alpha Name	Keyboard Name	Description	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.	*			158 159 285
		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		155
CLA	CLA	Clears Alpha register.	E		1	159

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

Clearing Functions (continued)

Alpha Name	Keyboard Name	Description	Stack	Flags	Bytes	Page
CLALMA		Clears first alarm whose message matches **target in Alpha register**.	E		2	258
CLALMX		Clears *n*th alarm listed by ALMCAT, ***n* specified X-register**.	E		2	259
CLD		Clears message from display.	E	50	1	318
CLFL		Clears file **named in Alpha register**.	E		2	213
CLKEYS		Clears all assignments on User keyboard.	E		2	168
CLP *label*		Clears the program in main memory containing specified global label.	E			286
CLRALMS		Clears all alarms.	E		2	258
CLRG		Clears all data storage registers in main memory.	E		1	202
CLRGX		Clears every *ii*th register from R_{bbb} through R_{eee} in main memory, ***bbb.eeeii* specified in X-register**.	E		2	202
CLΣ	CLΣ	Clears statistics registers.	E		1	190
CLST		Clears automatic memory stack.	E		1	183
CLX	CL*x*	Clears X-register.	D		1	159
DEL *nnn*		Deletes *nnn* program lines, starting with displayed line.	N			286
DELCHR		Deletes *n* characters **specified in X-register** from record in current text file, starting at current pointer.	E		2	225
DELREC		Deletes current record in current text file.	E		2	224
PCLPS		Clears program in main memory containing global label **specified in Alpha register** and all programs that follow it.	E		2	286
PURFL		Purges file **named in Alpha register**.	E		2	208

Stack/Data Register Functions

These functions manipulate the stack or the data storage registers, or take one of those registers as a parameter. (For functions that transfer data between data files in extended memory and the stack or main memory data registers, refer to "Extended Memory Functions" on page 425.) To interpret this table, refer to "Explanation of Table Entries" on page 414.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ASTO *nn*	ASTO *nn*	Copies six leftmost characters in Alpha register into R_{nn}.	I	E		2	200
ARCL *nn*	ARCL *nn*	Appends contents of R_{nn} to Alpha register.	I	E	28, 29, 36-41	2	200
CLRG		Clears all data storage registers.		E		1	202
CLRGX		Clears every *ii*th register from R_{bbb} through R_{eee}, ***bbb.eeeii* specified in X-register**.		E		2	202
CLΣ	CLΣ	Clears statistics registers		E		1	190
CLST		Clears automatic memory stack.		E		1	183
CLX	CLx	Clears X-register.		D		1	159
DSE *nn*		For *iiiii.fffcc* in R_{nn}, decrements *iiiii* by *cc* and skips next program line if *iiiii* − *cc* ⩽ *fff*.	I	E		2	306
ENTER↑	ENTER↑	Copies number in X-register into Y-register and lifts stack.		↑, D		1	175
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	306
LASTX	LASTx	Recalls number in LAST X register.		↑, E		1	179
PSIZE		Allocates *n* main memory registers to data storage, ***n* specified in X-register**.		E		2	199

Stack/Data Register Functions (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
R↑		Rolls up stack.		E		1	181
RCL *nn*	RCL *nn*	Recalls contents of R_{nn}.	I	↑, E		*	200
RDN	R↓	Rolls down stack.		E		1	181
REGMOVE		Copies contents of one block of *nnn* registers (starting with R_{sss}) to another block of *nnn* registers (starting with R_{ddd}), ***sss.dddnnn* specified in X-register**.		E		2	201
REGSWAP		Swaps contents of one block of *nnn* registers (starting with R_{sss}) with another block of *nnn* registers (starting with R_{ddd}), ***sss.dddnnn* specified in X-register**.		E		2	201
Σ+	Σ+	Accumulations for statistics.		L,D		1	191
Σ−	Σ−	Corrects statistics accumulations.		L,D		1	191
ΣREG *nn*		Assigns statistics registers to R_{nn} through R_{nn+5}.	I	E		2	190
ΣREG?		Returns address of first currently defined statistics register.		↑,E		2	190
SIZE *nnn*		Allocates *nnn* main memory registers for data storage.		E			199
SIZE?		Returns number of main memory registers currently allocated for data storage.		↑, E		2	199
ST+ *nn*	STO + *nn*	Adds number in X-register to number in R_{nn} and places result in R_{nn}.	I	E		2	201

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

Stack/Data Register Functions (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ST− *nn*	STO − *nn*	Subtracts number in X-register from number in R_{nn} and places result in R_{nn}.	I	E		2	201
ST* *nn*	STO × *nn*	Multiplies number in X-register by number in R_{nn} and places result in R_{nn}.	I	E		2	201
ST/ *nn*	STO ÷ *nn*	Divides number in X-register into number in R_{nn} and places result in R_{nn}.	I	E		2	201
STO *nn*	STO *nn*	Copies contents of X-register into R_{nn}.	I	E		*	200
VIEW *nn*	VIEW *nn*	Displays contents of R_{nn}.	I	E	21,50, 55	2	319
X<> *nn*		Exchanges contents of X-register with contents of R_{nn}.	I	E		2	201
X<>F		Exchanges number in X-register with status of flags 00 through 07.		E	00-07	2	295
X<>Y	x⇄y	Exchanges contents of X-register with contents of Y-register.		E		1	181

* If $00 \leq nn \leq 15$, 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 414.

Alpha Name	Keyboard Name	Description	Stack	Page
+	+	$y + x$.	L,↓,E	188
−	−	$y - x$.	L,↓,E	188
*	×	$y \times x$.	L,↓,E	188
/	÷	y / x.	L,↓,E	188
1/x	1/x	Reciprocal.	L,E	185
10↑X	10^x	Common exponential.	L,E	187
ABS		$\|x\|$ (Absolute value).	L,E	186
ACOS	COS^{-1}	Arc (inverse) cosine.	L,E	186
ASIN	SIN^{-1}	Arc (inverse) sine.	L,E	186
ATAN	TAN^{-1}	Arc (inverse) tangent.	L,E	186
CHS	CHS	Change sign.	E	185
COS	COS	Cosine.	L,E	186
D−R		Degrees to radians conversion.	L,E	187
DEC		Octal to decimal conversion.	L,E	187
E↑X	e^x	Natural exponential.	L,E	187
E↑X−1		Natural exponential for arguments close to zero.	L,E	187
FACT		$x!$ (Factorial).	L,E	185
FRC		Fractional part.	L,E	186
HMS		Decimal hours to hours-minutes-seconds conversion.	L,E	187
HMS+		Hours-minutes-seconds add.	L,↓,E	188
HMS−		Hours-minutes-seconds subtract.	L,↓,E	188
HR		Hours-minutes-seconds to decimal hours conversion.	L,E	187
INT		Integer part.	L,E	186
LN	LN	Natural logarithm.	L,E	187
LN1+X		Natural logarithm for arguments close to 1.	L,E	187

Numeric Functions (continued)

Alpha Name	Keyboard Name	Description	Stack	Page
LOG	LOG	Common logarithm.	L,E	187
MEAN		Means of accumulated *x*- and *y*-values.	L,E	192
MOD		*y* mod *x* (Remainder).	L,↓,E	190
OCT		Decimal to octal conversion.	L,E	187
P–R	P→R	Polar to rectangular conversion.	L,E	189
%	%	*x* percent of *y*.	L,E	188
%CH		Percent change from *y* to *x*.	L,E	188
PI	π	Pi (3.141592654).	↑,E	159
R–D		Radians to degrees conversion.	L,E	187
R–P	R→P	Rectangular to polar conversion.	L,E	189
RND		Round.	L,E	186
SDEV		Standard deviations of accumulated *x*- and *y*-values.	L,E	192
Σ+	Σ+	Accumulations for statistics.	L,D	191
Σ–	Σ–	Accumulations correction.	L,D	191
SIN	SIN	Sine.	L,E	186
SIGN		Sign of *x*.	L,E	186
SQRT	$\sqrt{x}$	Square root.	L,E	185
TAN	TAN	Tangent.	L,E	186
X↑2	x^2	Square.	L,E	185
Y↑X	y^x	*y* raised to the *x* power.	L,↓,E	189

Extended Memory Functions

All extended memory functions are programmable, requiring two bytes of program memory. To interpret this table, refer to "Explanation of Table Entries" on page 414 and to the following explanation of the special entries for this table.

File Types. Functions with an entry in this column act only on files of the indicated type.

P = Program file.

D = Data file.

A = Text (ASCII) file.

File Name. Functions with an entry in this column act on a file that is specified as indicated.

Yes = You must place the name of the desired file in the Alpha register.

OK = You may place the name of the desired file in the Alpha register, or you may clear the Alpha register to specify the current file.

No = This function acts only on the current file.

Pointer Used. Functions with an entry in this column act according to the current value of the indicated pointer.

RRR = register pointer (for data files).

rrr.ccc = record/character pointer (for text files).

rrr. = record pointer only (for text files).

Alpha Name	File Types	Description	File Name	Pointer Used	Stack	Flags	Page
APPCHR	A	Appends contents of Alpha register to end of current record.	No	*rrr.*	E		224
APPREC	A	Appends contents of Alpha register to end of current file as new record.	No		E		222
ARCLREC	A	Appends current record (starting at pointer to Alpha register.	No	*rrr.ccc*	E	17	226
ASROOM	A	Returns number of bytes available in current file.	No		↑,E		222

Extended Memory Functions (continued)

Alpha Name	File Types	Description	File Name	Pointer Used	Stack	Flags	Page
CLFL	A,D	Clears file **named in Alpha register.**	Yes		E		213
CRFLAS	A	Creates text file **named in Alpha register** containing *n* registers, ***n* specified in X-register**.	Yes		E		212
CRFLD	D	Creates data file **named in Alpha register** containing *n* registers, ***n* specified in X-register**.	Yes		E		211
DELCHR	A	Deletes *n* characters from record (starting at pointer), ***n* specified in X-register**.	No	*rrr.ccc*	E		225
DELREC	A	Deletes current record.	No	*rrr.*	E		224
ED	A	Activates Text Editor keyboard and display.	OK	*rrr.ccc*	E	26,28, 48	228
EMDIR		Lists directory of extended memory. Press R/S and then ← to terminate the listing and make the displayed file the current file.			E		206
		If allowed to finish, returns the number of available extended memory registers.			↑,E		
EMDIRX		Finds the *n*th file listed by EMDIR, ***n* specified in X-register**; returns file name to Alpha register and file type to X-register.			L,E		207
EMROOM		Returns number of registers available for a new file.			↑,E		208
FLSIZE	A,D,P	Returns number of registers in file.	OK		↑,E		208
GETAS	A	Copies mass storage file to extended memory file **named in Alpha register**.	Yes		E		227

Extended Memory Functions (continued)

Alpha Name	File Types	Description	File Name	Pointer Used	Stack	Flags	Page
GETP	P	Replaces last program in program memory with file **named in Alpha register**.	Yes		E	27	209
GETR	D	Copies corresponding registers from file to main memory.	OK		E		217
GETREC	A	Copies record (starting at pointer) to Alpha register.	No	*rrr.ccc*	E	17	226
GETRX	D	Copies file registers (starting at pointer) to R_{bbb} through R_{eee}, ***bbb.eee* specified in X-register**.	No	*RRR*	E		218
GETSUB	P	Copies file **named in Alpha register** to bottom of program memory.	Yes		E	27	209
GETX	D	Copies current register to X-register.	No	*RRR*	↑,E		220
INSCHR	A	Inserts contents of Alpha register, starting at character pointer.	No	*rrr.ccc*	E		224
INSREC	A	Inserts contents of Alpha register as new record.	No	*rrr.*	E		222
POSFL	A	Searches file for target **specified in Alpha register** and returns pointer value for match (−1 if no match).	No	*rrr.ccc*	↑,E		226
PURFL	A,D,P	Purges file **named in Alpha register.**	Yes		E		208
RCLPT	A,D	Returns pointer value for current file.	No	*rrr.ccc* *RRR*	↑,E		216
	P	Returns number of bytes in current program file.	No		↑,E		211

Extended Memory Functions (continued)

Alpha Name	File Types	Description	File Name	Pointer Used	Stack	Flags	Page
RCLPTA	A,D	Returns pointer value.	OK	*rrr.ccc RRR*	↑,E		216
	P	Returns number of bytes in program.	OK		↑,E		211
RESZFL	A,D	Resizes current file to *n* registers, ***n* specified in X-register**.	No		E		213
SAVEAS	A	Copies extended memory file **named in Alpha register** to mass storage file **named in Alpha register**.	Yes		E		227
SAVEP	P	Copies program **named in Alpha register** to file **named in Alpha register**.	Yes		E		208
SAVER	D	Copies all main memory registers to file.	OK		E		217
SAVERX	D	Copies contents of R_{bbb} through R_{eee} in main memory to file (starting with current file register), ***bbb.eee* specified in X-register**.	No	*RRR*	E		218
SAVEX	D	Copies number in X-register to current file register.	No	*RRR*	E		220
SEEKPT	A,D	Sets pointer for current file to number in X-register.	No		E		215
SEEKPTA	A,D	Sets pointer to number in X-register.	OK		E		215

Time Functions

To interpret this table, refer to "Explanation of Table Entries" on page 414.

Alpha Name	Description	Stack	Flags	Bytes	Page
ADATE	Appends number in X-register to Alpha register in current date format.	E	31, 36-39	2	243
ALMCAT	Lists alarms in chronological order. Can be executed manually as CATALOG 5.	E	31	2	255
ALMNOW	Activates the oldest past-due control or conditional alarm.	E		2	261
ATIME	Appends number in X-register to Alpha register in current time format.	E	36-39	2	240
ATIME24	Appends number in X-register to Alpha register in CLK24 format.	E	36-39	2	241
CLALMA	Clears first alarm whose message matches contents of Alpha register.	E		2	258
CLALMX	Clears *n*th alarm listed by ALMCAT, ***n* specified in X-register**.	E		2	259
CLK12	Selects 12-hour time display format.	E		2	239
CLK24	Selects 24-hour time display format.	E		2	239
CLKT	Selects time-only display for clock.	E		2	239
CLKTD	Selects time-and-date display for clock.	E		2	239
CLOCK	Displays clock.	E	31	2	238
	If executed by pressing ON, turns computer off and on, then displays clock.	E	12-26, 44-55		
CLRALMS	Clears all alarms.	E		2	258
CORRECT	Sets time and adjusts accuracy factor.	E		2	238
DATE	Returns number for current date.	↑,E	31	2	242
DATE+	Calculates new date from date in Y-register and number of days *n* in X-register. New date is later if *n* is positive, earlier if *n* is negative.	L,↓,E	31	2	244

Time Functions (continued)

Alpha Name	Description	Stack	Flags	Bytes	Page
DDAYS	Calculates the difference in days between dates in X- and Y-registers. Difference is positive if date in Y-register is earlier, negative if date in Y-register is later.	L,↓,E	31	2	244
DMY	Selects day-month-year date format.	E	31	2	242
DOW	Returns day-of-week number (0 = Sunday, 6 = Saturday) for date number in X-register.	L,E	31	2	244
HMS	Converts number in X-register from decimal hours format to hours-minutes-seconds format.	L,E		1	187
HMS+	Adds number in X-register to number in Y-register in hours-minutes-seconds format.	L,↓,E		1	188
HMS−	Subtracts number in X-register from number in Y-register in hours-minutes-seconds format.	L,↓,E		1	188
HR	Converts number in X-register from hours-minutes-seconds format to decimal hours format.	L,E		1	187
MDY	Selects month-day-year date format.	E	31	2	242
RCLAF	Recalls clock accuracy factor.	↑,E		2	376
RCLALM	Recalls XYZALM-type parameters for *n*th alarm listed by ALMCAT, ***n* specified in X-register**.	L,↑,E *	31	2	252
RCLSW	Returns stopwatch time.	↑,E		2	273
RUNSW	Runs stopwatch.	E		2	273
SETAF	Sets clock accuracy factor.	E		2	376
SETDATE	Sets clock to date **specified in X-register**.	E	31	2	242
SETIME	Sets clock to time **specified in X-register**.	E		2	237
SETSW	Sets stopwatch to starting time **specified in X-register**.	E		2	273
STOPSW	Stops running stopwatch.	E		2	273
SW	Activates Stopwatch keyboard and display.	E	26	2	266

* Copies the contents of the Y-register into the T-register, regardless of whether stck lift is enabled.

Time Functions (continued)

Alpha Name	Description	Stack	Flags	Bytes	Page
SWPT	Sets the stopwatch store and recall pointers by ***sss.rrr* in X-register**, and activates Stopwatch keyboard and display; returns current pointer values when Stopwatch keyboard is deactivated.	L,E	26	2	274
T+X	Adjusts clock time by **increment in X-register**.	E		2	238
TIME	Returns number for current time.	↑,E		2	240
XYZALM	Sets alarm for **time in X-register**, **date in Y-register**, **repeat interval in Z-register**, and **message or global label in Alpha register**.	E	31	2	250

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 414.

Alpha Name	Keyboard Name	Description	Flags	Page
	←	When input cue (_) is displayed, clears last digit or character entered; otherwise, clears displayed program line.		285
ASN	ASN	Assigns specified function or global label to specified key on User keyboard.		166
BST	BST	Displays preceding program line.		284
CAT *n*	CATALOG *n*	Executes catalog *n*, $1 \leqslant n \leqslant 6$.		170
CLP *label*		Clears program in main memory containing specified global label.		286
COPY *label*		Copies ROM program containing specified global label to program memory.		281

Editing Functions (continued)

Alpha Name	Keyboard Name	Description	Flags	Page
DEL *nnn*		Deletes *nnn* program lines, starting with displayed line.		286
	GTO .	Goes to specified line number or global label.		283
	GTO . .	Goes to bottom of program memory; packs program memory and creates null program.		281
ON		Selects continuous on (disables time-out).	44	155
PACK		Packs program memory.		198
SIZE *nnn*		Allocates *nnn* main memory registers for data storage.		199
SST	SST	Displays next program line.		284

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 414.

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	318
CLOCK		Stops program execution and displays the clock.		E	31	2	238
DSE *nn*		For *iiiii.fffcc* in R_{nn}, decrements *iiiii* by *cc* and skips next program line if *iiiii* − *cc* ⩽ *fff*.	I	E		2	306
END		Marks end of program.		E		3	301
FC? *nn*		Tests flag *nn* (00 ⩽ *nn* ⩽ 55) and skips next program line unless flag *nn* is clear.	I	E	*nn*	2	304

Functions That Direct Program Execution (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
FC?C *nn*		Tests flag *nn* ($00 \leq nn \leq 29$), clears flag *nn*, and then skips next program line unless flag *nn* was clear.	I	E	*nn*	2	304
FS? *nn*	FS? *nn*	Tests flag *nn* ($00 \leq nn \leq 55$) and skips next program line unless flag *nn* is set.	I	E	*nn*	2	304
FS?C *nn*		Tests flag *nn* ($00 \leq nn \leq 29$), clears flag *nn*, and then skips next program line unless flag was set.	I	E	*nn*	2	304
GETP		Replaces last program in memory with program file **named in Alpha register**. If last program calls new program (and is replaced), execution transfers to first line of new program.		E	27	2	209
GTO *label*	GTO *label*	Transfers execution to specified global, numeric, or local Alpha label.	I	E		*	300
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	306
LBL	LBL	Global, numeric, or local Alpha label.		E		†	299
OFF		Turns off the computer.		N	11-26 44-55	1	292
PROMPT		Displays contents of the Alpha register and stops execution.		E	50	1	314

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

† If $00 \leq nn \leq 14$, 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
RTN	RTN	Returns execution to line following XEQ instruction that called this subroutine.		E		1	301
STOP	R/S	Stops execution.		E		1	302
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	319
X = 0?	x = 0?	Skips next instruction unless number in X-register = 0.		E		1	304
X ≠ 0?		Skips next instruction unless number in X-register ≠ 0.		E		1	304
X < 0?		Skips next instruction unless number in X-register < 0.		E		1	304
X < = 0?		Skips next instruction unless number in X-register ≤ 0.		E		1	304
X > 0?		Skips next instruction unless number in X-register > 0.		E		1	304
X = Y?	x = y?	Skips next instruction unless contents of X-register = contents of Y-register.		E		1	304
X ≠ Y?		Skips next instruction unless contents of X-register ≠ contents of Y-register.		E		1	304
X < Y?		Skips next instruction unless number in X-register < number in Y-register.		E		1	304
X <= Y?	x ≤ y?	Skips next instruction unless number in X-register ≤ number in Y-register.		E		1	304
X > Y?	x > y?	Skips next instruction unless number in X-register > number in Y-register.		E		1	304

Functions That Direct Program Execution (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
X = NN?		Skips next instruction unless contents of X-register = contents of R_{nn}, ***nn* specified in Y-register**.		E		2	305
X ≠ NN?		Skips next instruction unless contents of X-register ≠ ontents of R_{nn}, ***nn* specified in Y-register**.		E		2	305
X < NN?		Skips next instruction unless contents of X-register < contents of R_{nn}, ***nn* specified in Y-register**.		E		2	305
X <= NN?		Skips next instruction unless contents of X-register ≤ contents of R_{nn}, ***nn* specified in Y-register**.		E		2	305
X > NN?		Skips next instruction unless contents of X-register > contents of R_{nn}, ***nn* specified in Y-register**.		E		2	305
X >= NN?		Skips next instruction unless contents of X-register ≥ contents of R_{nn}, ***nn* specified in Y-register**.		E		2	305
XEQ *label*	XEQ *label*	Calls specified global, numeric, or local Alpha label as subroutine. (If you specify a function, refer to the table entry for that function.)	I	E		*	301

* 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 414.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
	⊢	Appends subsequent characters to Alpha register.		N			159
ADATE		Appends number in X-register to Alpha register in current date format.		E	31, 36-39	2	243
ALENG		Returns number of characters in Alpha register.		↑,E		2	313
ANUM		Returns first digit string in Alpha register.		↑,E	22,28, 29	2	311
AOFF		Deactivates Alpha keyboard.		E	48	1	159
AON		Activates Alpha keyboard.		E	48	1	159
ARCL *nn*	ARCL *nn*	Appends contents of R_{nn} to Alpha register.	I	E	28,29, 36-41	2	200
ARCLREC		Appends record (starting at pointer) to Alpha register.		E	17	2	226
AROT		Rotates contents of Alpha register by *n* places, ***n* specified in X-register**. Rotates left for positive *n*, right for negative *n*.		E		2	313
ASHF		Shifts six leftmost characters out of the Alpha register.		E		1	200
ASTO *nn*	ASTO *nn*	Copies six leftmost characters in Alpha register into R_{nn}.	I	E		2	200
ATIME		Appends number in X-register to Alpha register in current time format.		E	36-39	2	240

Alpha Functions (continued)

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ATIME24		Appends number in X-register to Alpha register in CLK24 format.		E	36-39	2	241
ATOX		Shifts leftmost byte out of Alpha register and returns its decimal value to X-register.		↑, E		2	311
AVIEW	AVIEW	Displays contents of Alpha register.		E	21,50, 55	1	318
CLA	CLA	Clears Alpha register.		E		1	159
GETREC		Copies record (starting at pointer) to Alpha register.		E	17	2	226
POSA		Searches Alpha register for target **in X-register** and returns position of match (−1 if no match).		L, E		2	312
PROMPT		Displays contents of Alpha register and stops program execution.		E	21,50, 55	1	314
XTOA		Converts number in X-register to equivalent byte and appends it to Alpha register.		E		2	309

Interactive Functions

To interpret this table, refer to "Explanation of Table Entries" on page 414.

Alpha Name	Keyboard Name	Description	IND	Stack	Flags	Bytes	Page
ADV		Advances paper (if printer is present).		E	21,55	1	368
BEEP	BEEP	Sounds four tones.		E	26	1	319
GETKEY		Waits up to 10 seconds for a key to be pressed; returns key code (0 if no key is pressed).		↑,E		2	317
GETKEYX		Waits up to *SS.s* seconds for a key to be pressed, **SS.s specified in X-register**; returns keycode to Y-register and character code to X-register.		L,↑,E *	28,48	2	317
PROMPT		Displays contents of Alpha register and stops execution.		E	50	1	314
PSE		Delays execution for about one second.		E		1	315
TONE *n*		Sounds tone *n*, $0 \leqslant n \leqslant 9$.	I	E	26	2	319

* Copies the contents of Y- and Z-registers into the Z- and T-registers respectively, regardless of whether stack lift is enabled.

Indexes

Subject Index

Page numbers in **bold** type indicate primary references; page numbers in regular type indicate secondary references. Also, page numbers in *italic* type are in volume 1, while page numbers *not* in italic type are in volume 2.

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

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 three page references. The first one, in *italics*, is for volume 1. The second one is for volume 2. The third one, in **boldface**, is for the Function Tables, a summary in volume 2 of all functions.

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

00041-90492 English

Printed in Singapore 8/83