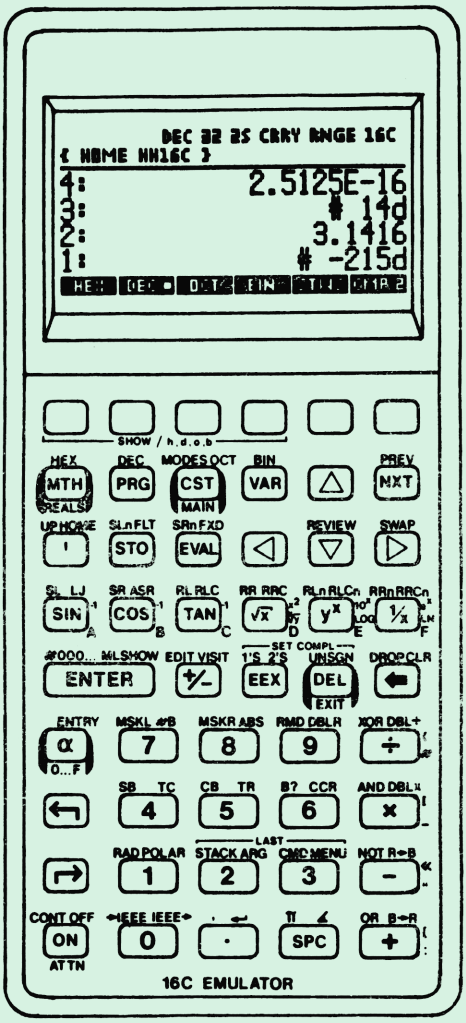
HP16C Emulator Library for the HP48S/SX



by
Jake Schwartz
and
Rick Grevelle

16C Emulator Registration Card

In order to be notified of program updates, documentation updates, or new products, please fill out the information below. Tear out this sheet and mail to:

Jake Schwartz
135 Saxby Terrace
Cherry Hill, New Jersey 08003-4606
USA

Product: **HP16C Emulator for the HP48S/SX**

Version: (Please circle one:) **Disk** **Card**

Serial Number: _____

Name: _____

Address: _____

_____

_____

_____

Comments: _____

_____

_____

_____

HP16C Emulator Library for the HP48S/SX

by
Jake Schwartz
and Rick Grevelle

Edition 1 - April, 1993 coinciding with software version 1.20

Notes and Acknowledgements

Back in the summer of 1990, as I was beginning to get comfortable with the HP48SX, my work at General Electric was demanding an increasing amount of time working with one's-complement octal integers. The Navy computers for which we were writing software hadn't yet caught up with the rest of the world as far as adopting two's-complement and hexadecimal formats - which seemed to be the standard everywhere else. As a result, my trusty 1982-vintage HP16C "computer scientist" calculator (being the only one before or since which could handle the job) remained close at hand. And, as it became inconvenient to carry both the HP16C and HP48SX at all times, I began to wonder if the '48 and RPL language might not provide an adequate platform onto which the various operations on signed octal integers could migrate.

By the fall of that year, I had managed to write a small group of RPL "user-code" objects which emulated the 16C's handling of all four integer bases in conjunction with both unsigned and signed numbers along with the four arithmetic functions and some bit manipulations. The idea was to create a situation where "16C mode" could be quickly entered, used and exited without adversely affecting any other HP48 conditions. However, due to the complexity of displaying negative decimal integers with a minus sign and the slowness of activating user keys which assigned all the HP16C functions to the keyboard, the thing never ran fast enough to be really useful.

By the Spring of 1991, a few people from the HP calculator user community had gotten wind of the 16C emulator project and offered both encouragement and suggestions on how to make it fly. However, no solutions came forth which could both integrate everything into a package that started as quickly as a single key-press and could function smooth enough to appear as seamless as the built-in HP48 applications.

Then, at the August 1991 HP Handheld Users' Conference in Corvallis, Oregon, I made a brief presentation on optimizing the HP48 software user interface and cited the abandoned 16C emulator code as an example. It was there that Rick Grevelle saw what had been attempted and thought about possibly helping make the project work. Beginning in early 1992, Rick became fully involved, applying his self-taught expertise in the inner-workings of the HP48 to this task. One by one, he was able to tackle virtually every obstacle that was encountered. As a result, we achieved the goal of coaxing

the HP48 to fully emulate the HP16C functionality without imposing on the user the burden of an unfriendly environment or complex labyrinth of menus. Without Rick's dedication, enthusiasm and knowhow, this project could not have been completed.

The 16C Emulator code was developed both on a PC and on the HP48 itself. The major development tools used were Hewlett-Packard's System-RPL Development Tools (available on EduCalc's HP48 Goodies Disk #4), Jan Brittenson's HP48 Machine Language Development Library ROM card and Brian Maguire's HP48 System-RPL Development Library.

Additional thanks are due to Richard Nelson for providing continuing encouragement all along the way. Jim Donnelly and Brian Maguire offered valuable suggestions at several points in the development. Our gratitude also goes out to Joseph Horn, Brian Walsh, Steve Thomas and Wlodek Mier-Jedrzejowicz for finding bugs, sharing ideas and pushing buttons. And finally, hearty thanks go out to Detlef Mueller for putting out some rather nasty fires in the code when time had just about run out. In addition, he implemented several significant features including the IEEE-format functions in System RPL, allowing VISITing of variables and turning on the intereactive stack in the emulator environment, and the MLSHOW function. Many thanks, guys.

Jake Schwartz
April, 1993

Table Of Contents

1. Introduction

A. Overview

The HP16C Emulator/Computer Science Library is a package which provides functionality for the HP48S/SX to perform integer arithmetic, bit manipulation, base conversion and more in either unsigned, one's-complement or two's-complement modes. Word sizes from 1 to 64 bits are supported. In addition to encompassing the full run-mode functionality of the Hewlett-Packard HP16C calculator, the library includes additional features to enhance the HP48 user interface. All functions may be immediately accessed quickly with the aid of a keyboard overlay and also may be found in the multi-page library-object soft-key menu. All status information (current word size, complement mode, base) is shown in the HP48's display status area. In addition, all of the functions of the library may be incorporated in users' programs.

The HP48 multi-level RPL stack and its accompanying commands (including **SWAP, DROP, CLR, LAST,** stack editing, etc.) are all available in "16C Mode." Entry of integers does not require the use of the pound-sign delimiter as a result of the library's context-sensitive parser. If a number consists solely of relevant digits for the current base, the number is interpreted as an integer; likewise if the number contains other digits or a decimal point and/or an exponent of ten, it is accepted as a real. Math with real numbers is supported as well with the HP48's row-4 scientific functions accessible by toggling into emulator "Math Mode." The hexadecimal keys A through F have been moved down from keyboard row 1 in the ALPHA plane to primary key positions in row 4 where they are much closer to the numeric key pad.

Conversion between integers and reals is supported via both the HP48's **R→B / B→R** function pair and the emulated versions of the HP16C's **FLOAT/FIXED*** function pair. The display shows integers in the selected base with or without leading zeros (in **BIN, OCT** or **HEX**) and correctly shows negative integers with a minus-sign (in **DEC**). Most regular HP48

* In the HP16C calculator, converting a floating-point real number to integer ("FIXED") format is performed by pressing the key corresponding to the desired integer base.

modes (such as display, angular, clock display on/off, multiline on/off, etc.) are adjustable through the HP48 **MODES** menu which remains active.

All keys whose functions are not permitted in the emulator environment are disabled, including illegal digit keys under the current base.

B. This Manual

This manual assumes that the user has a working knowledge of the HP48 calculator, its RPL stack operations and the functions in the **MTH BASE** menu which operate on integer numeric values.

C. Conventions Used

Throughout this manual, there are examples of HP48 keystroke sequences, problems and display representations. In order to differentiate between these items, three different type styles have been used. They are as follows:

Type of data:	Example of type style:
Numeric input	1.2345
HP48 function	**RCL ENTER STO**
HP48 LCD display information	# 12345d

2. Getting Started

A. Installing the Emulator

i. From Disk (at least 25K free RAM required):

The 16C Emulator Library may be loaded directly from the supplied floppy disk or from a PC hard drive. Insert the disk into the computer. A directory of the disk should show the following:

MUL8R.LIB	nnnnn	(16C Emulator Lib. file)
EXAMPLES.DIR	mmmm	(Programming examples)

If it is desired, the contents of the disk may be copied to the computer's internal hard drive and transferred to the calculator from there. Plug the HP48 serial interface cable into the computer and the calculator.

On the PC, start Kermit transfer protocol, set the baud rate to 9600 and type the SEND command:

```
C: KERMIT
C: SET BAUD 9600
C: SEND A:MUL8R.LIB        (or SEND MUL8R.LIB if
                            transferring from the current
                            directory in the hard drive)
```

Make sure the current HP48 VAR directory is the HOME directory. On the HP48, the I/O SETUP menu should indicate that "WIRE", "9600", and "BINARY" modes are active. At least 25K bytes of RAM should be free in order to download the library. (As a result, the presence of a RAM card would be useful.) From the I/O menu, press **RECV** and the emulator library should begin to be loaded into the calculator after a second or two. When the file transfer is complete, the serial cable may be disconnected.

In the HP48 VAR menu, the "MUL8" soft-key label should appear. Pressing the corresponding key should cause the display show the following in stack level 1:

1: Library 364: HP16...

Now, in order to preserve memory, purge the copy from the menu by pressing:

' [MUL8] ↩ PURGE .

This should delete the soft key but leave the library remaining on the stack. Press 0 **STO** to store the library into main memory in port 0. Now, turn the calculator off and back on again and the Emulator library will auto-attach. (The LCD will blink off and on.)

To confirm that the library is accessible, press ↩ **LIBRARY** and see "HP16C" as one of the soft-key labels in the LIBRARY menu.

ii. From Plug-in Card:

Turn the HP48 off and insert the 16C Emulator Library card into an open HP48 port. Turn the calculator back on. To confirm that the library is accessible, press ↩ **LIBRARY** and see "HP16C" as one of the soft-key labels in the LIBRARY menu.

B. Entering the Emulator Environment

After the library is installed and attached, the HP48 **LIBRARY** menu will include a key labeled **HP16C**. Pressing this key will display the first page of the HP16C Emulator Library menu as shown:

The **EMUL** key starts up the HP16C emulator environment (referred to as "HP16C Mode" from here on in) while the **ABOUT** key displays product

information about the library. The four arithmetic keys (and all subsequent keys on the seven other soft-key menu pages) are individual functions that are available on primary or shifted keys inside the emulator environment. These will all be described later in this manual.

Pressing **EMUL** activates HP16C mode in the calculator. Entering HP16C mode will produce a display similar to the one below.

Figure 1. Initial HP16C Emulator display.

In this example, the current base was decimal, the word size was 32 bits and the stack was empty before HP16C mode was entered. (These settings would have been created from the HP48's built-in **MTH BASE** menu functions.) Note the "16C" status message in the upper right of the display status area. In addition, the "32" shows the current binary word size, the "0S" message indicates that the current complement mode is "unsigned" and the "CRRY" and "RNGE" messages show that the emulator's carry and out-of-range (or "range" for short) flags are currently set. In addition, the first page of the soft-key menu shows that the current base is decimal (similar to the HP48 **MTH BASE** menu) and also shows that the current complement mode is "0" (or unsigned).

Each time the HP16C emulator is exited, the various modes are saved and restored again when the mode is re-entered. In addition, the status of leading-zeros display mode and math mode (both to be described later) are also saved and restored in a similar manner. These modes are retained in HP48 flags and thus require no user RAM memory. The various flags relevant to the Emulator are shown in the following table:

	Function	Flags	
Formerly unused system flags:	Complement Modes:	-13	-14
	Unsigned	clear	clear
	1's Complement	set	clear
	2's Complement	set	set
Already used system flags:	Current Word Size	-5 thru -10	
	Current Base	-11,	-12
	BIN	clear	set
	OCT	set	clear
	DEC	clear	clear
	HEX	set	set
User flags:	Math Mode	61 (on if set)	
	Leading-Zeros Display Mode	62 (on if set)	
	Carry Flag	63	
	Out-of-Range Flag	64	

C. Keyboard Operation

When 16C Mode is entered, the HP48 keyboard becomes redefined. The keyboard overlay supplied with the emulator is recommended to utilize the emulator functions more easily. Sometimes it is difficult to attach the overlay due to the six small tabs on the sides which must be inserted into the side slots on the HP48 keyboard. One recommendation would be to snip off some of these tabs with a sharp scissors. If all three tabs are removed from either the left or right side of the overlay, installation becomes a simple matter of inserting three tabs on the intact side and dropping the overlay into place.

The initial 16C mode keyboard layout is shown below:

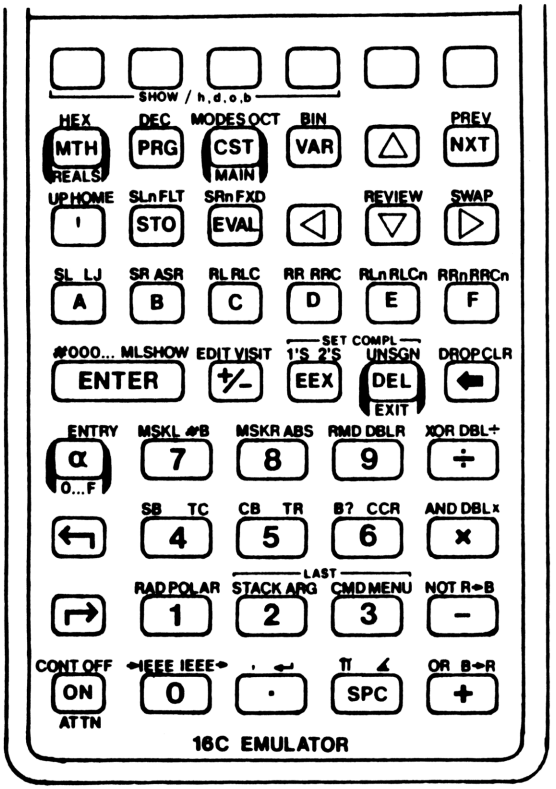


Figure 2. HP16C Emulator full keyboard.

Both the primary and shifted key definitions have been redefined in many cases. Most of the other primary keys have their definitions either retained or enhanced to operate in 16C mode. The keys in the numeric key pad are active based on the current base. If the base is binary, only keys 0 and 1 are allowed and pressing any of the others will result in an error beep tone. If the base is octal, only keys 0 through 7 are active. In decimal base, not only are keys 0 through 9 allowed, but the decimal point and **EEX** keys are also active so as to permit entry of real numbers as well as integers. If hexadecimal base is current,

not only are the 0 through 9 and decimal point and **EEX** keys active, but also the keys in row 4 primary positions act as hex digits **A** through **F** keys. (The relocation of the **A** through **F** keys from their original positions on the top row of the alpha-shifted plane down to the fourth row not only brings all the hex keys closer together and allows entry without shifts, but allows the emulator to use the top row for soft-key definitions at all times.) A picture of the 16C Mode active numeric keypad for each of the four integer bases is shown below:

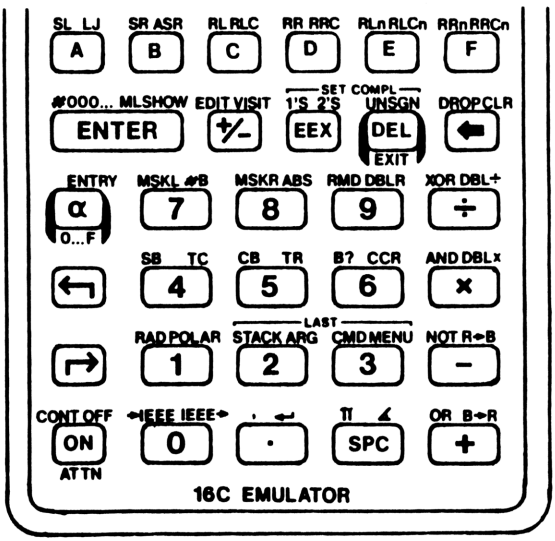


Figure 3. Keyboard layout for 16C mode with current base hexadecimal.

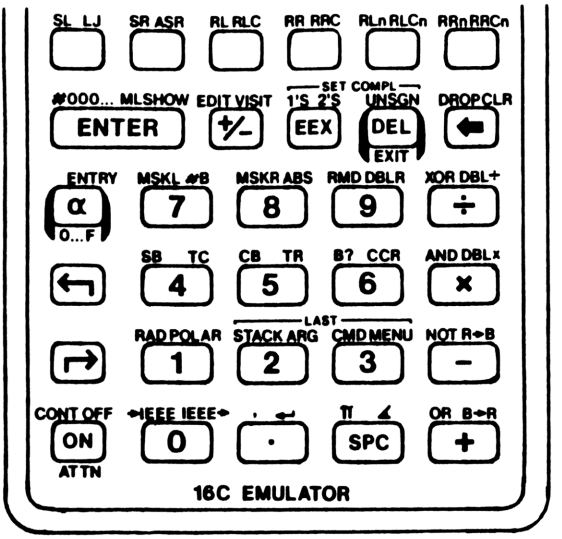


Figure 4. Keyboard layout for 16C mode with current base decimal.

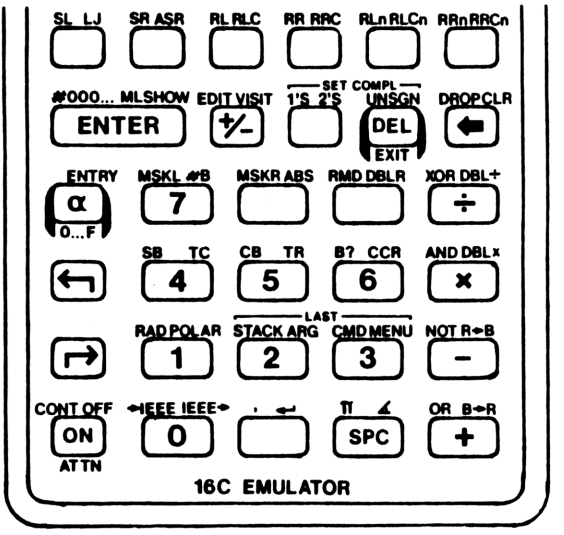


Figure 5. Keyboard layout for 16C mode with current base octal.

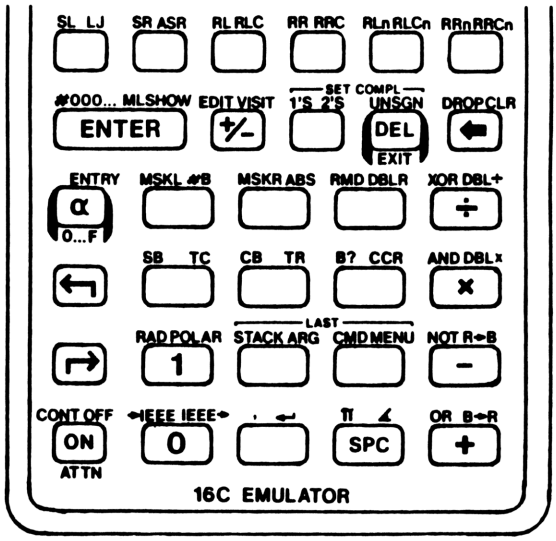


Figure 6. Keyboard layout for 16C mode with current base binary.

Keys like **NXT**, the four "arrow" keys, **ENTER**, **+/-**, **EEX**, back-arrow, the arithmetic keys and **ATTN** all retain their original definitions and operate in the context of the emulator. The ' (tick), **STO**, **EVAL**, and **VAR** keys are also active during 16C mode.

i. The Primary Key Definitions

Four primary keys - **MTH**, **CST**, **DEL** and α have completely new definitions.

Pressing the **DEL** key exits the emulator, returning to the HP16C library menu.

The α key activates all the hexadecimal (0 through F) digit keys along with the decimal point and **EEX** keys to allow entry of reals or numbers in any base at any time. (Note that when the "fraction mark" is set to a comma through the **MODES** menu, the decimal-point key produces a comma like the regular HP48 behavior.) This key still turns on the alpha display annunciator and stays on only as long as the normal α state does - until

ENTER or a function causing an automatic evaluation is pressed. In addition, α may be locked with two consecutive presses or by setting system flag -60.

The **CST** key returns the top row keys to their "main-row" functionality (similar to the HP48 action of the blue-shifted **NXT** key press).

The **MTH** key toggles the HP48 row-4 keys and most of their left- and right-shifted definitions back and forth from their 16C-mode definitions to their original functionality, providing limited scientific calculation functions for real numbers while in the 16C emulator. In addition, the numeric keypad reverts to that in 16C mode for decimal base: keys 0 through 9 plus the decimal point and **EEX** are active. Also, the HP48 object delimiters are activated in their original left- and right-shifted arithmetic key positions. When in **MTH** mode, the 16C status message in the top right of the LCD is changed to "MTH". A picture of the **MTH** mode keyboard is shown below.

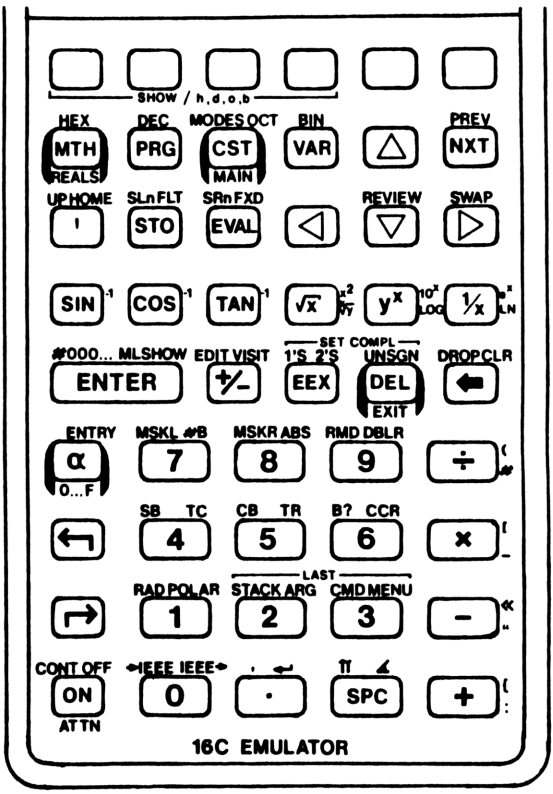


Figure 7. HP16C Emulator MTH Mode keyboard.

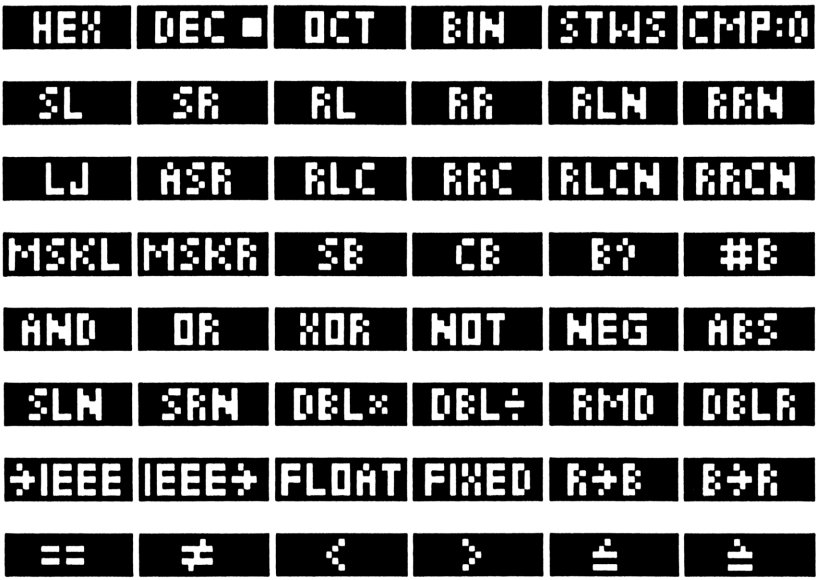
ii. The Shifted Key Definitions

During 16C mode, several shifted key definitions remain intact. These are **MODES, PREV, UP, HOME, REVIEW, SWAP, EDIT, VISIT, DROP, CLR, ENTRY, RAD, POLAR, LAST STACK, LAST ARG, LAST CMD, LAST MENU, CONT, OFF,** comma (,), carriage return (↵), π and ∠. The newly-defined functionality will be described throughout this manual.

In most places on the keyboard where the overlay only shows a single shifted (orange or blue) definition (such as with the **VAR** or **SPC** keys), this means that the other shifted key definition is undefined. If the overlay shows no shifted functionality (such as with the **up-arrow** key), then both shifted positions are undefined.

D. Soft-Key Menu Functions

The HP16C-Mode top-row soft keys are dynamically labeled in the bottom row of the display like the normal HP48. The initial (or "main") row of definitions are the ones which appear whenever the emulator is entered. There are seven additional rows of six key definitions following the main row which may be accessed by repeatedly pressing the **NXT** key. The entire soft-key menu is shown below:



The menu includes most of the functions found in the HP48's **MTH BASE** multi-page menu, along with all the integer functions from the HP16C calculator. These functions, which all also appear elsewhere on the redefined 16C-mode keyboard, are here in the soft-key menu as a backup measure in case the keyboard overlay is not available.

E. Returning to the "Main" Menu Page

When any row of key definitions beyond the main (emulator's initial) row (or a different menu such as **MODES** or **VAR**) is displayed, the user may immediately return to the main row by pressing the **CST** key.

F. HP48 Functions Remaining Active

The **REVIEW** operation shows a summary of the six functions in the current page of the 16C-mode soft-key menu. The regular HP48 stack-manipulation functions **SWAP**, **DROP**, and **CLR** are still active. When there is no command line, **SWAP** and **DROP** are also accessible as primary functions on the right-arrow and back-arrow keys, respectively. The **RAD** key switches between radians and degrees mode (or between radians and grads mode if **GRADS** was selected from the **MODES** menu). The **POLAR** key switches between rectangular and polar coordinates mode (with polar mode being either cylindrical or spherical mode, depending on which has been chosen in the **MODES** menu). The **MODES** menu remains accessible for changing display notation, turning the clock on or off, adjusting the radix and so forth.

The **PRG** key remains active in order to allow access to the **PRG STK**, **PRG OBJ**, **PRG BRCH** and **PRG TEST** menus underneath. (Access to the **PRG CTRL** and **PRG DSPL** menus have been disabled.)

The **VAR** key provides the user access to HP48 objects currently residing in RAM memory. In conjunction with the soft keys in this menu, the normal left-shifted and right-shifted **STO** and **RCL** capabilities to and from user objects has been retained.

The four **LAST** functions (**LAST STACK**, **LAST ARG**, **LAST CMD**, **LAST MENU**) from the regular HP48 keyboard are retained in 16C mode in their original left- and right-shifted positions above the 2 and 3 keys.

The **EDIT** and **VISIT** functions operate on stack entries in the way that the normal HP48 does. If no command line is active, **EDIT** copies the object in stack level 1 into the command line for editing and turns on the **EDIT** menu. If a command line is already present, **EDIT** simply activates the **EDIT** menu. The **VISIT** function allows editing of stack level n if value n is placed in stack level 1 or into the command line; or normal editing of user-created objects from the **VAR** menu.

The **ENTRY** function allows the command line to be filled with a sequence of commands, delaying evaluation until **ENTER** is pressed. If no command line exists, **ENTRY** causes the command line to be started.

The **OFF** key turns the calculator off from inside the 16C emulator. When the HP48 is turned back on, the calculator will be in the same mode as when it was before it was shut off.

The arrow keys assume most of their normal roles while in 16C mode. If no command line is active, the right arrow performs a **SWAP**. The down arrow begins editing the value in stack level 1. The up arrow begins the interactive stack. All four arrow keys work correctly to move the cursor around during an editing session. The blue-shifted arrow keys still move the cursor to the far edges of the data in the command line. Since there is no access to the graphics (PICT) display, the left arrow key is disabled if no command line is present.

The **UP**, **HOME**, comma (,), carriage-return (↵), π, ∠, tick ('), **STO** and **EVAL** keys are also active in the emulator mode. They function in their usual manner.

G. Exiting the Emulator

The **DEL** key acts as the exit key to leave the emulator. All current settings in the emulator are retained and are reactivated when re-entered. (Note, that the word size and current base settings are saved in flags which are also used for the same settings made in the **MTH BASE** menu. If these are changed outside the emulator, the new values will be reflected when the emulator mode is re-entered.)

3. Number and Display Control

A. Display Annunciators

The top row of the display status area shows up to eight different messages. The table below shows these messages moving from right to left in the display:

Annunciator	Function
RAD/GRAD	Angular mode
R∠Z / R∠∠	Polar coordinate mode
HEX / DEC / OCT / BIN	Current integer base
1 or 2-digit number	Current word size
0S / 1S / 2S	Current complement mode
CRRY	Carry flag on
RNGE	Out-of-range flag on
16C / MTH / PRG	16C mode/Math mode/ Delayed-eval entry mode

The second row of the status area remains unchanged in emulator mode. The current path followed by the clock display (if turned on) is shown.

B. Number Base Modes

The four base modes of the emulator work just like those in the HP48 **MTH BASE** menu. However, in order to aid the user in numeric data entry of integers, the digit keys that are not relevant to the current base are disabled. Thus:

If current base is:	The following digits are active:
BIN (base 2)	0 and 1
OCT (base 8)	0 through 7
DEC (base 10)	0 through 9
HEX (base 16)	0 through 9 plus A through F

Unlike the regular HP48 where in order to enter binary integers one must precede the numbers by the pound-sign delimiter, the HP16C emulator does not have this requirement. Simply entering the numeric value and pressing **ENTER** will place the integer onto the stack.

The current base (saved in system flags -11 and -12) outside the emulator is carried over to 16C emulator mode for consistency.

It should also be noted that in decimal and hexadecimal bases, the decimal point and **EEX** keys are also active to allow entry of real numbers as well as integers. See section 5 ("Real Numbers and Alpha Mode") for more information.

i. A Second Set of BASE Keys

In addition to the **HEX**, **DEC**, **OCT**, and **BIN** keys on the emulator's MAIN soft-key page, the base functions have been duplicated in the right-shifted positions on the second row of keys directly below their counterparts. This allows base manipulation while the soft-key menu displays another emulator page or perhaps another menu altogether. Due to space considerations, the keyboard overlay does not show the blue **DEC** function label in the shifted second row; however its functionality is indeed present in the emulator.

C. Temporary Display ("SHOW")

If one wishes to temporarily view the values in the stack in a base other than the currently selected base, the **SHOW** functions may be utilized. Pressing left-shift followed by a base key in the emulator "MAIN" soft-key menu displays the stack values in the selected base for a short period of time, after which the system returns to the original base. After left shift is pressed, if the

key of the selected base to be shown is continually held down, the display will remain in the new base until the key is released. Attempting to do a show base with the current base will act as if no key had been pressed.

D. Entry in any Base

Sometimes the requirement exists to enter an integer in a different base than the current base. The right-shifted base keys in the "main" soft-key menu page append lower-case letters h, d, o or b to the numeric value in the command line, thus allowing the calculator to interpret the number in the desired base. For example, if the current base is decimal and the value octal 3776 is to be entered, the keystrokes 3776 **OCT ENTER** will enter "3776o" into the command line, convert the number to the current base and place it on the stack as "#2046d."

Despite the fact that the keys corresponding to irrelevant digits in the current base are disabled in 16C mode, a user might wish to enter a value in a base which requires the "illegal" digits. Pressing the α key will temporarily enable all of the digit keys 0 through F to allow entry in any base. (The α annunciator at the top of the display will also turn on.) Like the regular HP48, ALPHA mode remains on for only one keystroke if it is not locked. By setting system flag -60 before pressing αor by pressing the α key twice, it will remain locked until the entire command line is entered. Of course when **ENTER** is pressed, ALPHA mode will terminate, causing the "illegal" digit keys revert to their inactive state. The α display annunciator will also darken. Also note that while ALPHA is on, the rest of the entire 16C mode keyboard is active with the same functionality (with the exception of the shifted arithmetic keys, which take on the original HP48 object delimiter functionality). Note that the normal three ALPHA planes of characters are disabled throughout the 16C emulator.

Example:

While in 16C mode with the current base binary, a user wishes to enter the integer 9AF2 hexadecimal. System flag -60 is not currently on. The key sequence would be:

α α 9 A F 2 ↵ **HEX ENTER**

The stack level one value will then show "#1001101011110010b." In order to confirm that the correct value was entered, pressing **SHOW HEX** will briefly convert level one to #9AF2h.

E. Complement Modes and Unsigned Mode

In addition to the regular HP48's functions associated with binary integers which only operate on unsigned values, the 16C emulator also provides 1's Complement and 2's Complement signed modes. In the binary representation of signed numbers, the most significant bit is the sign bit; which is cleared for positive and set for negative values. When the current base is decimal, the emulator (like the 16C itself), displays negative numbers with a minus sign in front of the digits. Since the HP48 shows integers with the leading pound-sign delimiter to distinguish them from reals, the minus sign is placed after the # and just before the leading digit of the negative integer.

To control the complement modes while in the emulator, two methods have been provided. First, in the main page of the soft-key menu, the sixth key labeled "**CMP:n**" where n = 0, 1 or 2 is a three-way toggle between unsigned, ones complement and twos complement modes. Each time this soft key is pressed, the key label will change to reflect the current complement mode. Secondly, the shifted functions above the **EEX** and **DEL** keys in row 5 of the HP48's keyboard provide direct paths to any of the three complement modes. By either method, when the complement mode is changed, the message in the top row of the LCD status area changes to reflect the current complement mode, showing "**0S**", "**1S**" or "**2S**" (as well as the digit being updated in the soft-key label).

In 1's complement mode, negating the value in stack level 1 (by pressing the **+/-** key) will perform a 1's complement on the value. This is done by complementing all bits in the value.

In 2's complement mode, the **+/-** key will take the 2's complement of the value in stack level 1. All the bits will be complemented followed by adding 1 to the value.

In unsigned mode, no sign bit is used and all the bits in the number contribute to its magnitude. Negating an unsigned number is actually meaningless, however pressing the **+/-** key in this mode will result in the 2's complement of the stack level-1 value. In addition, the out-of-range flag will be set (and the **RNGE** annunciator will turn on in the display) as a reminder that the true result is a negative value outside the range of the current unsigned mode.

The table below depicts the effects of the complement mode on the representation of numbers in decimal in the display. A word size of 5 bits is used in this example.

Binary Base	Octal Base	Hex Base	Decimal		
			Unsigned	1's compl.	2's compl.
01111	17	F	15	15	15
01110	16	E	14	14	14
01101	15	D	13	13	13
01100	14	C	12	12	12
01011	13	B	11	11	11
01010	12	A	10	10	10
01001	11	9	9	9	9
01000	10	8	8	8	8
00111	7	7	7	7	7
00110	6	6	6	6	6
00101	5	5	5	5	5
00100	4	4	4	4	4
00011	3	3	3	3	3
00010	2	2	2	2	2
00001	1	1	1	1	1
00000	0	0	0	0	0
11111	37	1F	31	-0	-1
11110	36	1E	30	-1	-2
11101	35	1D	29	-2	-3
11100	34	1C	28	-3	-4
11011	33	1B	27	-4	-5
11010	32	1A	26	-5	-6
11001	31	19	25	-6	-7
11000	30	18	24	-7	-8
10111	27	17	23	-8	-9
10110	26	16	22	-9	-10
10101	25	15	21	-10	-11
10100	24	14	20	-11	-12
10011	23	13	19	-12	-13
10010	22	12	18	-13	-14
10001	21	11	17	-14	-15
10000	20	10	16	-15	-16

It becomes evident here that the 1's complement representation in decimal provides an equal number of positive and negative numbers, with both zero and minus zero. With 2's complement mode, however, there is only one zero but an additional negative number exists at the highest magnitude.

Note: Since the display representation of negatively-signed decimal integers using a minus sign is solely a feature of the 16C emulator, these numbers will appear differently when the 16C mode is exited.

Example:

In 16C Mode, with a word size of 12 bits, the current base decimal and 2's complement mode on, enter the numbers -255, -122 and -6. Compare the HP48 display to that when the emulator is exited.

The keystroke sequence is:	The display shows:
DEC ↷ 2's 1 2 **STWS**	
2 5 5 **+/- ENTER**	**DEC 12 2S 16C**
1 2 2 **+/- ENTER**	3: # -255d
	2: # -122d
6 **+/- ENTER**	1: # -6d

Now, exit the emulator and observe the display:

DEL	3: # 3841d 2: # 3974d 1: # 4090d

Re-entering the emulator will return to the original numerical representation:

EMUL	DEC 12 2S 16C
	3: # -255d 2: # -122d 1: # -6d

F. Word Size Control

The HP16C emulator (like the HP48 itself) operates on integers whose word sizes range from 1 to 64 bits long. The word size in the HP48, maintained in system flags -5 through -10, is carried over to 16C mode just like the current base. Unlike the requirement outside the emulator to press **RCWS** (from the **MTH BASE** menu) in order to ascertain the current word size, the 16C emulator displays this information (in base ten) at all times in the top line of the display status area (between the current base and the current complement mode). In order to change the current word size to n bits, the value n is entered into the stack and **STWS** ("store word size") is pressed from the main page in 16C mode. This updates the number in the LCD status area as well.

Either integers or reals may be used as the input argument to the **STWS** command. (A discussion on entry of real numbers in 16C mode may be found in chapter 5.) For real arguments, the value is rounded to the nearest positive whole number. Arguments which are negative cause the word size to be set to 1 bit. Arguments above 64 set the word size to 64. If the input argument is zero, the word size is set to 64 as well. For integer arguments, values of 64 and over or zero cause the word size to be set at 64. A negative integer argument causes the word size to equal the absolute value of the input value.

If the input **STWS** argument is:	The word size will be set at:
#-1 or lower	I input value I
#0	64 bits
#1 to #64	1 to 64 bits
#65 or higher	64 bits
0.0 or negative	64 bits
1.0 - 63.499999	1 to 63 bits
64.0 or higher	64 bits

A current word size less than 8 bits will limit the size of the integer value that may be used to set a new word size. However, entering either integer 0 as an input argument (to yield a word size of 64 bits) or a real argument (to set the word size to any desired value) will circumvent the problem.

Note 1: Changing the word size might not maintain numerically equivalent values as they are displayed in the stack. Since moving to a larger word size adds bits (which are set to zero) onto the left-hand end of the word, displayed negative values in decimal can change to positive, since the most significant bit (which was previously set) would now be within the word itself. Likewise, moving to a smaller word size will cause only the new smaller (least-significant) portion of the data values to be displayed (in any base). Immediately following the reduction of the word size, if the word size is restored, the values will return to their original magnitudes. However this will not be the case if other operations (such as arithmetic) are performed on the stack first. This causes only the current word size portion of the data to be used and the remaining portion discarded.

Example:

Enter various integers and change the word size to see the displayed results. Begin in 1's complement mode with a word size of 32 and a base of hexadecimal.
f CLR HEX 32. **STWS** **g 1's**

Keystrokes:	Display shows:	Comments:
1AFE6321 **ENTER**	# 1AFE6321h	Starting value
DEC	# 452879137d	Convert to decimal
24 **STWS**	# -105694d	Reduce word size, note value and sign changes
HEX	# FE6321h	Back to HEX, value changes
32 **f DEC STWS**	# 1AFE6321h	Back to original word size
DEC 24 **STWS**	# -105694d	Reduce word size again
0 **+**	# -105694d	Add zero, result unchanged
32 **STWS**	# 16671521d	Restore word size again, value changed
HEX	# FE6321h	Value remains smaller; upper 8 bits cleared

Note 2: In either of the signed modes, it is possible that the entry of a positive number larger than the largest positive integer that can be represented in the current word size will result in a negative number in the stack. For a current word size of n, the value that ends up displayed in the stack is the lowest n bits of the integer in the command line. Thus, if the most significant bit becomes set, the value will be displayed as negative.

If the word size is currently 3 bits or less, it is possible to enter a single digit that is legal in the current base, but is too large for the given word size. Again, in this case, the lowest n bits of the digit will end up displayed in the stack.

Example:

Starting in 2's complement mode, decimal base and a word size of 4 bits, enter various large numbers and observe the results.

Keystrokes:	Stack shows:	Comments:
f CLR DEC f 2's 4 STWS	(empty)	Initial state
12000 ENTER	# 0d	Lowest 4 bits clear
HEX	# 0h	Still clear
DROP DEC 12345 ENTER	# -7d	Most significant bit is set
BIN	# 1001b	Shows the bits
α α 12340 f DEC ENTER	# 100b	Lock ALPHA to enable all digits for decimal data entry
DEC	# 4d	See it in decimal
9 ENTER	# -7d	1 digit > allowed value

G. Leading-Zeros Display Mode

Often it is helpful for integers in the stack to be displayed with zeros to the left of the highest nonzero digit up to the entire word size. This can aid in determining whether the most significant bit is set or clear when the current base is other than decimal (where a leading set bit in signed mode would cause the value to be displayed with a minus sign). Choosing the **#000...** function on the left-shifted **ENTER** key toggles into and out of leading-zeros display mode. This mode is preserved when exiting and re-entering the emulator and is always suppressed when the current base is decimal.

H. The Multi-line SHOW Function (MLSHOW)

If the current word size is set high enough such that integers in the stack exceed sixteen digits in length, the HP48 cannot display the entire contents of those integers at one time. At the right-hand end of the number an ellipsis (...) appears, implying that additional digits are hidden. Even if the integer is edited, the entire value cannot be displayed at one time. Moreover, using the arrow keys to view the hidden portions of the integer can sometimes be misleading.

The emulator's multi-line show (**MLSHOW**) function (on the right-shifted **ENTER** key) assists in simultaneously viewing all digits of large integers. When the **MLSHOW** key is pressed, the normal stack display is cleared and replaced with a view of the stack level-one value alone. The value is displayed right-justified with up to two groups of 8 digits per display line, starting at the bottom. The entire 4-line display may be filled by a 64-bit binary integer with 8 groups of 8 binary bits.

Note that while the command line is active or a program is being entered, both the left- and right-shifted **ENTER** key positions generate a **DUP** function, just like the normal HP48.

Example:

Initial conditions: Base: HEX Word size: 64 Complement Mode: 2's
Leading Zero's Mode OFF

HEX 64. STWS 2's CLR

Keystrokes	Display Shows	Comments
25 ENTER MLSHOW	25	
123456789ABCDEF0 ENTER	# 123456789ABCDEF0h	A 16-digit number
MLSHOW	12345678 9ABCDEF0	Two groups of 8 digits
DEC MLSHOW	131 17684674 63790320	19 digits long in decimal
+/- MLSHOW	-131 17684674 63790320	Supports negative values
OCT MLSHOW	166713 52303545 20620420	22 digits in octal
BIN MLSHOW	11101101 11001011 10101001 10000111 01100101 01000011 00100001 00010000	64 digits in binary
CLR OCT 125 ENTER MLSHOW	000000 00000000 00000125	Supports leading-zeros mode
α α 17. STWS BIN	# 0000000000101010...	Full binary value hidden
MLSHOW	0 00000000 01010101	

I. Command-Line Editing / ENTRY mode

Like in the regular HP48, a command line is started either by typing directly into the HP48 keyboard while in 16C mode or by **EDIT**ing or **VISIT**ing values in the stack. When a command line is being edited, the up- and down-arrow keys operate normally, moving the cursor within the displayed text. The blue-shifted left-and right-arrows will also move the cursor to the extreme left and right ends of the command line as well. If a multiline object is present, the up and down arrows change the current line in which the cursor is positioned and the blue-shifted up and down-arrows move to the first and last lines of the command line object, respectively.

If a command line is activated by either **EDIT** (to edit stack level 1) or **VISIT** (with a stack-level number as an input parameter), the **EDIT** menu is automatically enabled. With a command line activated by typing in the HP48 keyboard, the **EDIT** menu may always be activated by pressing ↩ **EDIT** at any time.

i. Operation of the Change-sign (+/-) Key in the Command Line

Like with the regular HP48, while in 16C mode the change-sign key can have varying effects depending on the position of the cursor in the command line. If a single number has been entered, pressing change-sign will place a minus sign in front of positive values and change the minus sign on negative values to a plus sign. If multiple values are entered into the command line separated by one or more spaces, change-sign will affect the value nearest to the cursor. To negate a value, simply move the cursor over to that value.

If a command line is achieved by editing a stack value, the change-sign key can have different effects depending whether the value is a real or an integer. If the value is a real, pressing change-sign will change the sign of the value; on the other hand if the value is an integer, the change-sign key will negate the value and place it back onto the stack. Another effect of the change-sign key is the way the 16C emulator mode parser interprets the resulting number in the command line. If the current base is decimal, a negative value in the command line will be read in as an integer and entered onto the stack

as such. If the current base is binary or octal, a value in the command line with a minus sign will be interpreted as a negative real.

In hex, if the negative value in the command line contains only digits between 0 and 9, it will be entered as a real. If the value begins with a digit between 0 and 9 but finished with a digit or digits between A and F, or only contains digits between A and F, pressing the change-sign will negate the value and enter it onto the stack. If the value begins with a digit between A and F and continues with other digits, the effects of the change-sign key become more complicated. Sometimes changing the sign of a mix of digits between 0 and F can cause the minus sign to be embedded inside the number. This will generate an Invalid Syntax error if **ENTER** is pressed.

The table below summarizes the behavior of the change-sign key.

If the current base is:	And the number in the command line is:	Pressing +/- with the number in the command line will cause:	Then pressing **ENTER** will:
DEC	A string of digits WITH a decimal point and with or without a sign	A minus (plus) sign to appear in front of a positive (negative) integer	Accept the number as a real
DEC	An integer with a leading "#" delimiter	The number to be negated and entered onto stack	N / A
BIN or OCT	A string of digits with or without a dec. pt. and with or without a sign	A minus (plus) sign to appear in front of a positive (negative) number	Accept the number as a real
BIN or OCT	An integer with a leading "#" delimiter	The number to be negated and entered onto stack	N / A
HEX	A string of digits (0-9 only) with or without a dec. pt. and with or without a sign	A minus (plus) sign to appear in front of a positive (negative) number	Accept the number as a real
HEX	A string of digits starting with 0-9 and continuing with A-F digits	The number to be negated and entered onto the stack	N / A
HEX	A number containing only A-F digits	The number to be negated and entered onto stack	N / A

If the current base is:	And the number in the command line is:	Pressing +/- with the number in the command line will cause:	Then pressing **ENTER** will:
HEX	A number starting with A-F digits and continuing with 0-F digits ending in A-F	The number to be negated and entered onto the stack	N / A
HEX	A number starting with any digits and continuing with 0-F digits ending in 0-9	A minus sign to appear between the last A-F digit and the 0-9 digits after	Cause an "Invalid Syntax" error
HEX	An integer with a leading "#" delimiter	The number to be negated and entered onto stack	N / A

Example:

Enter numbers into the command line under various conditions, press change-sign and observe the results.

Keystrokes	Display Shows	Comments
↩ MODES STD CST		Initial conditions
DEC 32 STWS 1.234	1.234	Initial value
+/-	-1.234	Negated
ENTER	-1.234	Entered onto stack
↪ CLR 1234 ENTER ↩ EDIT	# 1234d	Editing a decimal integer
+/-	# -1234d	Negates and enters
↪ CLR OCT 1234	1234	Enter octal number
+/-	-1234	Negated
+/-	+1234	Negated again
ENTER	1234	Accepted as a real
↪ CLR 1234 ENTER ↩ EDIT	# 1234o	Editing an octal integer
+/-	# 37777776544o	Negates and enters
↪ CLR HEX 1234	1234	Enter hex number
+/-	-1234	Negated
ENTER	-1234	Accepted as a real
↪ CLR 1234FED ENTER ↩ EDIT	# 1234FEDh	Editing hex number

Keystrokes	Display Shows	Comments
+/-	# FEDCB013h	Negates and enters
← CLR 12FED56	12FED56	Hex number ending in 0-9 digit, leaving cursor just after the last digit
+/-	12FED-56	Minus sign before last set of decimal digits
ATTN ATTN FED	FED	Hex number with A-F digits only
+/-	# FFFFF013h	Negates and enters

The moral to the story is to be careful when the change sign key is pressed with data in the command line. Consult section 5 for more information on dealing with reals in the 16C emulator.

J. Access to the MODES Menu From Inside the Emulator

While in 16C mode, it might become necessary to change one or more of the HP48 modes which are accessible from the **MODES** menu. A modified version of the **MODES** menu is available while the 16C emulator is active. The functions appear in the following order:

The beeper on/off toggle key has been replaced by the fraction mark key, which has been moved up from the last page. The connect-plotted-points toggle key on the second page has been replaced by the leading-zeros display mode ("LZ") toggle key. Page three is unchanged and page four has been eliminated, since the base selections are an integral part of the emulator.

4. Integer Arithmetic and Bit Manipulation Functions

The HP16C emulator contains virtually all of the integer arithmetic and bit manipulation functions of the HP16C, plus a few more.

A. Carry and Out-of-Range Conditions

Several 16C-mode functions are subject to carry and out-of-range conditions. Like in the original HP16C, the carry and out-of-range conditions are saved in flags (being user flags 63 and 64 respectively). In the display status area will appear "**CRRY**" and "**RNGE**" if the conditions are present. In addition, the keyboard functions "**TC**" (Toggle Carry flag), "**TR**" (Toggle Range flag) and "**CCR**" (Clear Carry and Range flags) have been provided in order to manually adjust these conditions.

The list below shows the functions which set or clear the carry flag (and also adjust the **CRRY** display annunciator) when they are performed on integer arguments.

SL	**RL**	**RLn**	+ (carry)
SR	**RLC**	**RLCn**	- (borrow)
SLn	**RR**	**RRn**	+ (nonzero remainder)
SRn	**RRC**	**RRCn**	**DBL+** (nonzero remainder)
ASR			$\sqrt{x}$ (nonzero remainder)

Example: The following additions set and clear the carry condition.

DEC 16 **STWS** **2's**

Keystrokes	Display Shows	Comments
HEX FFFF **ENTER**	# FFFFh	
2 +	# 1h / **CRRY**	Carry set
4 +	# 5h	Carry cleared

The out-of-range condition occurs when the correct result of an operation cannot be represented in the current word size and complement mode. The following functions adjust the out-of-range condition (and thus adjust the **RNGE** annunciator in the display) whenever they are performed:

+ - × + ABS +/- DBL× DBL÷

Additionally, the **FLOAT**, **→IEEE** and **IEEE→** functions affect **RNGE**.

When a result is out of range, the lowest bits of the correct value which fit in the current word size are returned. For addition and multiplication of integers, the sign bit of the full answer will be retained in the most significant bit in the current word size.

Example:

Multiply the sets of two numbers and observe the results.

DEC 8 **STWS** ◀ **2's**

Keystrokes	Display Shows	Comments
11 **ENTER**	# 11d	
50 ×	# 38d / **RNGE**	(Out-of-Range annunciator turns on)
◀ **CLR** ◀ **CCR**		
25 **ENTER**	# 25d	
6 **+/-** ×	# -106d / **RNGE**	(Negative result Out-of-Range enabled)

In addition to the **TC**, **TR** and **CCR** functions on the keyboard, the flag tests **CRRY?** (is the carry flag set?) and **RNGE?** (is the out-of-range flat set?) are available in the emulator library for use in program objects. The functions return 0 or 1 like the built-in HP48 **FS?** test. Also, the library provides **SETC** (set carry), **CLRC** (clear carry), **SETR** (set range) and **CLRR** (clear range) functions.

B. Integer Arithmetic Functions

i. Addition, Subtraction, Multiplication and Division

The operations addition, subtraction, multiplication and division may be performed on two integers in any of the four number bases. Division results in an integer quotient with the fractional part truncated. All integer arithmetic operations except for multiplication will affect the condition of both the carry and out-of-range flags (and corresponding display annunciators). Multiplication affects the out-of-range flag only.

The results of integer arithmetic are affected by the current word size and complement mode.

Example:

Initial conditions: Base: HEX Word size: 32 Complement Mode: 2's

HEX 32. **STWS** ↻ **2's** ↻ **CCR**

Keystrokes	Display Shows	Comments
32AB **ENTER**	# 32ABh	
14 +	# 288h / CRRY	Nonzero remainder: carry flag set
20 +	# 2A8h	Carry is cleared
OCT	# 1250o	Switch to octal
2751 -	#37777776277o/ CRRY	Result is negative; carry set
7 **STWS**	# 77o / CRRY	Reduce word size

Example continued:

Keystrokes	Display Shows	Comments
↩ TC	# 77o	Clear carry
70 ×	# 10o / RNGE	Large result sets out-of-range
DEC ↩ TR	# 8d	Switch to decimal, clear range
11 +	# 0d / CRRY	Carry set due to remainder

Note: Division of a non-zero integer by integer zero or integer zero by integer zero causes an Undefined Result error. This is not the case in the regular HP48; division of an integer by integer zero outside the emulator returns integer zero.

a. Mixed Integer and Real Arithmetic Arguments

Like on the regular HP48, arithmetic may be performed with an integer and a real argument. The result will always be of the integer type, with the real value having been converted to integer (as if the real-to-binary function had been performed on the number first). This conversion results in a rounding of the real to the nearest integer. If the complement mode is unsigned, any negative reals involved in the arithmetic operation will be converted to zero first.

Example:

Initial conditions: Base: HEX Word size: 32 Complement Mode: 2's

HEX 32. **STWS** ↩ **2's** ↩ **CCR**

Keystrokes	Display Shows	Comments
1A **ENTER**	# 1Ah	
2. ×	# 34h	Result is integer in hex
1.1 +	# 35h	Fractional part rounded off
2.9 -	# 32h	Value rounds to integer 3 before being subtracted
2.1 +	# 19h	Effective division by two

b. Addition and Subtraction in 1's Complement Mode

While in 2's complement or unsigned mode, the result of addition or subtraction represents simply the sum or difference of the bit patterns of the two numbers being combined. However, in 1's complement mode, the result of addition may be affected by the occurrence of a carry and, likewise, the result of subtraction may be affected by the occurrence of a borrow. If a carry out of the most significant bit occurs, a 1 is added to the result. If a borrow into the most significant bit occurs, 1 is subtracted from the result. In both carry and borrow situations, the carry flag (and **CRRY** annunciator) is set.

Examples:

Initial conditions: Base: BIN word size: 4 Complement Mode: 1's Leading Zero's Mode on

BIN α 4 **STWS** ↩ **1'S** ↩ **#000...**

Keystrokes	Display Shows:	Comments:
1110 **ENTER**	# 1110b	Carry Situation:
		Decimal Binary
1110 +	#1101b/ CRRY	
		111
DEC ↻ CLR ↻ TC		-1 1110
		+ (-1) + 1110
1 **+/-** **ENTER**	# -1d	-2_{10} 1100_2
1 **+/-** +	# -2d / CRRY	+ 1
		1101_2
BIN ↻ CLR ↻ TC		
1100 **ENTER**	# 1100b	No Carry Situation:
11 +	# 1111b	Decimal Binary
DEC ↻ CLR		-3 1100
		+ 3 + 0011
3 **+/-** **ENTER**	# -3d	-0_{10} 1111_2
3 +	# -0d	
BIN ↻ CLR		
11 **ENTER**	# 0011b	Borrow Situation:
100 -	# 1110b/CRRY	Decimal Binary
DEC ↻CLR ↻TC		3 $^{1}_{1}$ $\emptyset$011
		-4 -0100
3 **ENTER**	# 3d	-1_{10} -1
4 -	#1d / CRRY	1110_2
BIN ↻ CLR ↻ TC		

Keystrokes	Display Shows:	Comments:
110 **ENTER**	#0110 b	No Borrow Situation:
101 -	# 0001 b	Decimal: Binary:
DEC ↷ **CLR**		6 $0^{0}1^{1}10$ -5 -0101 1_{10} 0001_2
6 **ENTER**	# 6d	
5 -	# 1d	

c. The Carry Flag During Addition

Whenever a binary addition results in a carry beyond the most significant bit, the carry flag and **CRRY** annunciator will be set. If an addition does not result in a carry, the flag (and annunciator) will be cleared. This is true for all complement modes.

Example:

Initial conditions: Base: BIN Word size: 4 Complement Mode: 2's
Leading Zero's Mode on

BIN α 4 **STWS** ↷ **2's** ↶ **#000...**

Keystrokes	Display Shows	Comments	
1010 **ENTER**	# 1010b	Decimal	Binary
1100 +	# 0110b / RNGE , CRRY	-6	^{1} 1010
		+(-4)	+1100
1 +	# 0111b	-------	-------
		6_{10}	0110_2
		(CRRY, RNGE set)	
DEC ↷ **CLR**			
		6	0110
6 **+/- ENTER**	# -6d	+1	+0001
		-------	-------
4 **+/-** +	# 6d / RNGE, CRRY	7_{10}	0111_2
1 +	# 7d	(CRRY, RNGE cleared)	

d. The Carry Flag During Subtraction

When a binary subtraction results in a borrow into the most significant bit, the carry flag (and **CRRY** annunciator) will be set. If a borrow does not occur, the carry will be cleared.

Example:

Initial conditions: Base: BIN Word size: 4 Complement Mode: 2's
Leading Zero's Mode on

BIN α 4 **STWS** ↷ **2's** ↶ **#000...**

Keystrokes	Display Shows	Comments	
1010 **ENTER**	# 1010b	Decimal	Binary
1100 -	# 1110b/CRRY	-6	$^{0}_{1}$1010
		-(-4)	−1100
DEC ↷ CLR ↷ TC		------	----------
		-2_{10}	1110_2
6 **+/- ENTER**	# 6d		
		(CRRY set)	
4 **+/-** -	# -2d / CRRY		
BIN ↷ CLR		Decimal	Binary
110 **ENTER**	# 0110b/CRRY	6	$^{0}_{1}$0110
		-1	−0001
1 -	# 0101b	-------	----------
		5_{10}	0101_2
DEC ↷ CLR ↷ TC			
6 **ENTER**	# 6d / CRRY	Turn on carry flag	
1 -	# 5d	(CRRY cleared)	

e. The Out-of-Range Flag During Arithmetic

The out-of-range flag will be set whenever arithmetic results cannot be represented in the current word size and complement mode. The **RNGE** display annunciator accompanies this flag. For integer division, this occurs only in 2's complement mode when the largest possible negative number (100000...) is divided by -1.

Example:

Initial conditions: Base: BIN Word size: 4 Complement Mode: 2's

BIN α 4 STWS 2's

Keystrokes	Display Shows	Comments
110 **ENTER**	# 111b	6
		+5
101 +	# 1011b / RNGE	------
		-5_{10}

f. Arithmetic with Other HP48 Objects

In addition to arithmetic with integers and reals, the 16C Emulator will support normal arithmetic functions with any other HP48 object types. Using the delimiters (which are accessible via the **MTH** mode), complex numbers, lists, vectors, etc. may be entered and combined. Also, if the emulator is entered with these objects already on the stack, they may be combined using the arithmetic keys.

ii. Remainder Following Division and RMD

During integer division, only the integer portion of the result is returned to the stack. If the actual remainder is non-zero, the carry flag (and **CRRY** annunciator) is set. The carry flag is cleared if the remainder is zero.

If the value of the remainder is desired, use the **RMD** function instead of **÷**. This function calculates

|stack-level-2| MOD |stack-level-1|.

The sign of the result will match the sign of the dividend, originally in level 2.

Example:

Initial conditions: Base: DEC Word size: 16 Complement Mode: 2's

DEC 16. **STWS 2's**

Keystrokes	Display Shows	Comments
7 +/- ENTER	# -7d	Division example
3 +	# -2d / CRRY	Integer quotient of 7/3 Remainder causes carry
2 +	# -1d	No remainder; carry clear
↩ CLR		
7 +/- ENTER	# -7d	RMD example
3 ↪ RMD	# -1d	Remainder of -7/3

iii. Square Root, y-to-the-x and x-squared

The square root and y-to-the-x functions, which are accessible on primary keys on row 4 in emulator **MTH** mode will operate properly with integer input arguments, yielding integer results. For the square root function, if the fractional portion of the result is non-zero, the carry flag will be set; otherwise carry is cleared.

The X-squared function, which is also accessible via emulator **MTH** mode (on the left-shifted row-4 key), also accepts integer arguments and returns an integer result.

For results which cannot be represented in the current word size, the lower portion of the integer will be retained.

All three aforementioned functions will process any input object type that the HP48 would normally allow.

iv. Negative Numbers and Complementing

a. Changing Signs

The **+/-** function (change sign) will change the sign of the stack level-1 value, forming the (1's or 2's) complement. If the initial value is the largest possible negative number in 2's complement mode, the only effect of pressing **+/-** will be to set the out-of-range flag (and to turn on the **RNGE** annunciator).

If the emulator is currently in unsigned mode, pressing the **+/-** key forms the 2's complement of the level-1 value. This also causes the out-of-range flag to be set as a reminder that negative numbers are outside the range of unsigned mode.

To enter a negative integer, press **+/-** after entering all the numeric digits (just like in the regular HP48).

b. Absolute Value of Integers

Pressing **→ ABS** converts the integer in stack level one to its absolute value, forming the 1's or 2's complement of a negative number. No change in the number results if the calculator is in unsigned mode or if the number is already positive.

If the stack holds the largest possible negative number in 2's complement mode, the only effect of **ABS** will be to set the out-of-range flag and turn on the **RNGE** annunciator.

The **ABS** function performs the normal absolute value function if the stack level-one value is a real number or any other HP48 object type the normally can be processed.

C. Logical Operations on Integers

The logical (Boolean) operations **NOT, OR, AND** exclusive OR (**XOR**) perform a bit-by-bit analysis of the binary integer argument(s). The dyadic functions **OR, AND** and **XOR** operate on the bits in corresponding positions of

the integers in stack levels 1 and 2. The monadic function **NOT** acts only upon the level 1 value.

i. The NOT Function

The **NOT** function inverts the values of all bits in the binary integer in stack level 1. This is equivalent to performing a **+/-** (change-sign) while in 1's complement mode. Only the stack level-1 value is affected.

Example:

Initial conditions: Base: BIN Word size: 16 Complement Mode: 2's Leading-Zero's Mode

BIN α α 16 **STWS** ↻ **2's** ↻ **CLR** ↺ **#000...**

Keystrokes	Display Shows	Comments
1111 **ENTER**	# 0000000000001111b	
↺ **NOT**	# 1111111111110000b	All bits are inverted

ii. The AND function

The **AND** function (also known as the logical product) performs the logical **AND** on each pair of corresponding bits in the integers in the lowest two stack levels. Each resulting bit becomes a 1 only if both corresponding input operand bits are 1; otherwise it becomes 0.

Example:

Initial conditions: Base: BIN Word size: 16 Complement Mode: 2's

BIN α α 16 **STWS** ↻ **2's** ↻ **CLR**

Keystrokes	Display Shows	Comments
1111000 **ENTER**	# 1111000b	Bits remain set only where both input arguments had bits set (just one bit)
11110001111  **AND**	# 1000b	

iii. The OR Function

The **OR** function (also known as the logical sum) compares each corresponding bit in the two lowest stack levels. Each resulting bit is 0 only if both input argument bits are 0's.

Example:

Initial conditions: Base: BIN Word size: 16 Complement Mode: 2's

BIN α α 16 **STWS** **2's** **CLR**

Keystrokes	Display Shows	Commands
10101 **ENTER**	# 10101b	Resulting zero bits represent where corresponding bits in inputs are both zero
10011 **OR**	# 10111b	

iv. The Exclusive OR (XOR) Function

The **XOR** function (also known as the logical difference) checks the corresponding bits in the integers in stack levels 1 and 2 and yields a 1 only if the bits are different.

Example:

Initial conditions: Base: BIN Word size: 16 Complement Mode: 2's

BIN α α 16 **STWS** **2's** **CLR**

Keystrokes	Display Shows	Comments
100111000 **ENTER**	# 100111000b	Positions where resulting bits are set mean input bit values differ there
101010101 **XOR**	# 1101101b	

D. Shifting and Rotating Bits

The bits of an integer may be moved left or right by performing shifting and/or rotating operations. The fate of the bit moved off the end of the integer and the value of the bit entering the vacated bit position depend upon the type of shift or rotate performed.

The carry flag is adjusted by any shift or rotate function except for the **LJ** (left-justify) function.

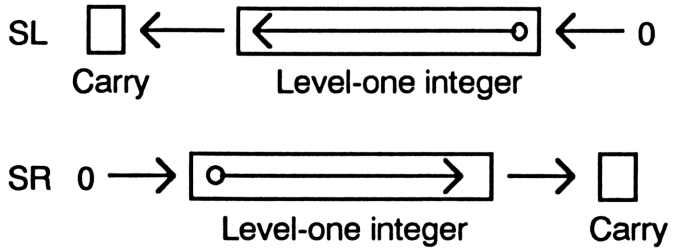
i. Shifting Bits

The 16C emulator can perform two types of shifts on the stack level-1 integer: a logical shift or an arithmetic shift. The latter preserves the sign bit. Also, the level-1 integer contents may be left-justified.

a. Logical Shifts **SL** and **SR**

Pressing **SL** (shift left) or **SR** (shift right) moves all the bits of the integer word in the bottom stack level by one bit to the left or right. A bit shifted out of the integer is shifted into the carry bit position, replacing the

previous state of the carry bit (flag). The new bits generated at the opposite end of the integer are always zeros.



b. Shifting More Than One Bit at a Time

The 16C Emulator also contains the functions **SLn** (shift left by n bits) and **SRn** (shift right by n bits). In these functions, the number of bits by which the integer in stack level two is shifted correponds to the value of the number in level one. The level-one value of n may be either integer or real. If the value of n is an integer, the level-two integer will be shifted by the number of bits corresponding to the absolute value of level one. If the value of n is real, the number of bits by which the level-two value is shifted will be the nearest integer to the level-one real number. In addition, if the value of n is a negative real, the level-two integer will be shifted by the magnitude of the level-one value IN THE OPPOSITE DIRECTION. No matter whether it is real or integer, the valid range for n is within plus or minus the current word size. Outside this range the **SLn** or **SRn** functions will yield a "Bad Argument Value" error.

The status of the carry flag following **SLn** or **SRn** is the same as if **SL** or **SR** were performed n times. Thus, if the FINAL bit shift causes a 1 bit to be shifted out of the integer value, the carry flag (and **CRRY** annunciator) will be set. Otherwise, the carry flag will be cleared.

Example:

Initial conditions: Base: BIN Word size: 16 Complement Mode: 2's
Display mode: STD

BIN α α 16 STWS 2's CLR

Keystrokes	Display Shows	Comments
1100 **ENTER**	# 1100b	
10 **SLn**	# 110000b	Value shifted left two bits
100 **SRn**	# 11b	Shifted right by four bits
10 **SRn**	# 0b/CRRY	Shift right by 2, carry set
CCR		Start over
1 **ENTER**	# 1b	Set least significant bit
10 **ENTER +/-**	# 1111111111111110b	Enter integer minus two
SLn	# 100b	Shifted left by two
α α 1.1 **ENTER**	1.1	n = real 1.1
SRn	# 10b	Shifted right by one
α α 1.9 **ENTER**	1.9	n = real 1.9
SLn	# 1000b	Shifted left by two
α 4 **+/- ENTER**	-4	n = real 4
SLn	# 0b / CRRY	Shifted right by four, carry set

c. Left-Justify (**LJ**)

To left-justify a bit pattern with its word size, press **LJ**. The resulting left-justified integer will be placed in stack level two and the count of the number of bits necessary to left-justify the word will go into level one. The carry flag is not affected by **LJ**.

Example: Left justify the binary value 10101 in a word size of eleven.

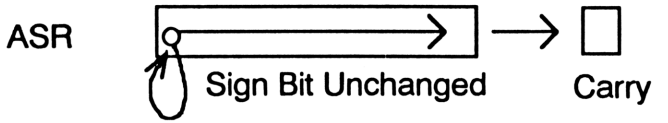
Initial conditions: Base: BIN Word size: 11 Complement Mode: 2's

BIN α α 11. **STWS 2's CLR**

Keystrokes	Display Shows	Comments
10101 **ENTER**	# 10101b	
LJ	2: # 10101000000b 1: # 110b	Value was shifted 6 bits to the left

d. Arithmetic Shift Right (**ASR**)

Pressing **ASR** will move the contents of the integer in stack level one a single bit to the right like **SR**. However, instead of placing a zero into the new place at the left end of the word (in the sign bit position), the sign bit is regenerated. (In unsigned mode where there is no sign bit, **ASR** operates exactly like **SR**.) The carry flag is set if a 1 is shifted out to the right of the integer and cleared if a 0 is shifted out.



Example: Divide -44 by 4 by shifting the value twice to the right.

Initial conditions: Base: DEC Word size: 8 Complement Mode: 2's

DEC 8 STWS 2's CLR CCR

Keystrokes	Display Shows	Comments
44 **+/- ENTER**	# -44d	Original value
SHOW BIN	# 11010100b	Display the individual bits
ASR	# -22d	Shift right once
SHOW BIN	# 11101010b	Bits shifted, sign maintained
ASR	# -11d	Shift right again
SHOW BIN	# 11110101b	Sign maintained again

ii. Rotating Bits

There are three types of rotation functions in the 16C emulator, encompassing eight different functions:

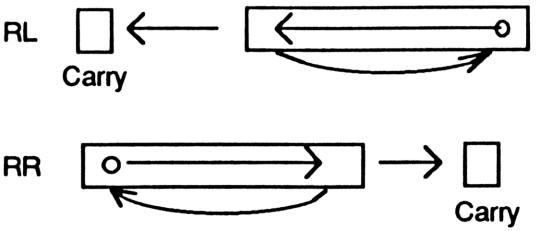
- Rotate left and right (**RL, RR**)

- Rotate left and right through the carry bit (**RLC, RRC**)

- Rotate n places (**RLn, RRn, RLCn, RRCn**)

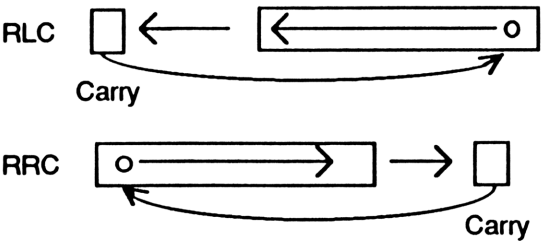
a. Rotation

The **RL** and **RR** functions cause the contents of the integer in stack level one to rotate (or circularly shift) one bit to the left or right. If a bit is shifted out of a word, it re-enters at the other end. The carry flag is set if a 1 bit is rotated around either end, and is cleared if a zero is rotated around either end.



b. Rotation Through the Carry Bit

The **RLC** (rotate left through carry) and **RRC** (rotate right through carry) functions respectively load the leftmost or rightmost bit of a level-one integer into the carry bit, and move the carry bit into the other end of the word.



c. Rotating More Than One Bit at a Time

The functions **RLn, RRn, RLCn** and **RRCn** allow rotation of the level-one integer by a multiple number of bits. In these functions, the number of bits by which the integer in stack level two will be rotated will correspond to the value in level one. The level-one value of n may be either integer or real. If the value of n is an integer, the level-two integer will be rotated by the number of bits corresponding to the absolute value of level one. If the value of n is real, the number of bits that the level-two value is rotated will be the nearest integer to the real number. In addition, if the value of n in level one is a negative real, the level-two integer will be rotated by the magnitude of n IN THE OPPOSITE DIRECTION. No matter whether it is real or integer, the valid range for n is

within plus or minus the current word size. Outside this range the rotate-by-n functions will yield a "Bad Argument Value" error.

The status of the carry flag is the same as if **RL, RR, RLC** or **RRC** were performed n times. Thus, if the FINAL bit rotation causes a 1 bit to be rotated out of the integer value, the carry flag (and **CRRY** annunciator) will be set. Otherwise, the carry flag will be cleared.

Example:

Initial conditions: Base: BIN Word size: 8 Complement Mode: 2's

BIN α α 8 STWS ↷ 2's ↷ CLR

Keystrokes	Display Shows	Comments
11000 **ENTER**	# 11000b	
α 4 ↷ **RRCn**	# 1b / CRRY	Rotated right 4 bits
α 2 ↷ **RRCn**	# 11000000b	Right two more: carry cleared
α 2 **+/-** ↷ **RRCn**	# 1b / CRRY	Rotated left 2 bits: carry set again
10 ↷ **RLCn**	# 110b	Left two bits again
α α 2.9 ↷ **RRCn**	# 11000000b	Rotated 3 to the right

It should be noted that while the 16C Emulator does not include the HP48's rotate and shift by a byte (**RLB, RRB, SLB** and **SRB**) functions, these can easily be duplicated by performing 8 **RLn**, 8 **RRn**, 8 **SLn** or 8 **SRn** respectively.

E. Bit Manipulation

i. Setting, Clearing and Testing Bits

Individual bits in an integer may be set to 1 or cleared to 0 using the **SB** (set bit) and **CB** (clear bit) functions. In addition to these bit functions, any bit may be tested for the presence of a 1 with the **B?** function.

To set, clear or test a specific bit in an integer, the integer must be in placed stack level two with the bit number in level one. Bit numbers are counted from right-to-left, beginning with zero for the least significant bit. The bit number may be either an integer or a real. Reals will be rounded to the nearest integer value. The legal range for the bit number parameter is plus or minus the current word size. Outside this range, the bit function will return a Bad Argument Value error. Within the legal range, the value for the bit number is taken as the absolute value of the level-one number.

When either **SB** or **CB** is executed, the updated integer value is returned to level one of the stack. The **B?** function operates like the other relational tests in the HP48: A 1 ("true") is returned if the bit is set and a 0 ("false") returned if the bit is clear.

Example:

Initial conditions: Base: BIN Word size: 8 Complement Mode: 2's

BIN α 8 **STWS** **2's** **CLR**

Keystrokes	Display Shows	Comments
11000 **ENTER**	# 11000b	
10 **SB**	# 11100b	Setting bit 2
100 **CB**	# 1100b	Clearing bit 4
α 3 **B?**	1	Testing bit 3

ii. Masking

The **MSKL** (mask left) and **MSKR** (mask right) functions create left- or right-justified strings of 1 bits. The magnitude of the number in stack level one is used to determine how many 1's will comprise the mask. Upon execution, **MSKL** or **MSKR** places the mask pattern into stack level one.

A mask may be created as large as the current word size. To create a mask which is in the middle of the field of a number, use one of the shift functions in conjunction with **MSKL** or **MSKR**.

Example: Extract the second most significant 8 bits from the 32-bit hex value 3ABC7101 and right-justify these bits.

Initial conditions: Base: HEX Word size: 32 Complement Mode: 2's

HEX 32. **STWS** ↷ **2's** ↷ **CLR**

Keystrokes	Display Shows	Comments
3ABC7101 **ENTER**	1: # 3ABC7101h	
	2: # 3ABC7101h	
8 ↶ **MSKL**	1: # FF000000h	Create 8-bit mask
	2: # 3ABC7101h	
8 ↶ **SRn**	1: # FF0000h	Shift the mask to right
↶ **AND**	1: # BC0000h	Extract the bits
16. ↶ **SRn**	1: # BCh	Shift them right

iii. Bit Summation

Pressing ↷ **#B** (number of bits) sums the set bits in the stack level-one integer and returns that value to level one. Note that this function is called "no.B" outside the emulator environment. This is due to the fact that the

HP48 would interpret "#B" to mean a hex integer value of B if entered in a program or command line.

F. "Double" Functions (DBL×, DBL÷, DBLR)

The 16C emulator (like the real HP16C) provides three "double" functions: **DBL×** (double multiply), **DBL÷** (double divide) and **DBLR** (double remainder). These functions perform the exact calculation of a product double the current word size and the exact calculation of a quotient and remainder from a dividend of double word size.

In order to obtain meaningful double numbers as results in hexadecimal and octal base modes, the word boundary (which is based on the current number of bits in the word size) must not "split" a digit. Therefore it is advised to specify a compatible word size; i.e., a multiple of four for base HEX and a multiple of three for OCT base. Due to the aforementioned word-boundary effects, there is little chance that the results of the double functions will be meaningful in decimal base mode.

i. Double Multiply

The **DBL×** function multiplies two single-word integers in stack levels one and two and returns a double-word result in the same two stack levels. The result is right-justified with the least significant bits returned in level two and the most significant bits returned in level one. A stack diagram during the double-multiply operation is shown below:

multiplicand	2: # yyyyyy	→	2: # ...yxyxyxy lower-order bits
multiplier	1: # xxxxxx	**DBL×**	1: # xyxyxyx... higher-order bits

Example:

Initial conditions: Base: BIN Word size: 7 Complement Mode: 2's

BIN α 7 STWS ◀ 2's ◀ CLR

Keystrokes	Display Shows	Comments
10110 **ENTER**	# 10110b	Start with 7-bit word size
11110 **↩ DBLx**	2: # 10100b 1: # 101b	Result in 14-bit double word
↩ CLR α α 14 **STWS**		Start over with 14-bit size
10110 **ENTER**	# 10110b	
11110 **×**	# 1010010100b <upper><lower>	Full result matches above

Unlike the original HP16C, where only the lowest stack level is visible in its one-line display, the HP48 allows the user to view both halves of the double-word result simultaneously.

ii. Double Divide

The **DBL÷** function computes the quotient of a dividend of double-word size in stack levels two and three (with the most significant bits in level two) divided by a single-word divisor in stack level one. The single-word result is placed in stack level one.

An Overflow Error occurs if the answer cannot be represented in a single word size. The carry flag (and **CRRY** annunciator) is set if the remainder is not equal to zero. The stack contents during the double divide operation are:

dividend lower half	3: # ...yzyzyz	→	
dividend upper half	2: # yzyzyz...	**DBL÷**	1: # (yz/x)... quotient
divisor	1: # xxxxxxxxx		

Example: Divide 1256 by 51 in binary with word size 14 and 2's complement mode and then double-divide the same values with word size of 7 to compare the results.

Initial conditions: Base: DEC Word size: 14 Complement Mode: 2's

DEC 14 **STWS** ↩ **2's** ↩ **CLR**

Keystrokes	Display Shows	Comments
1256 **ENTER** 51 **ENTER**	2: # 1256d 1: # 51d	Enter initial values in decimal
BIN	2: # 10011101000b 1: # 110011b	Convert to see the binary representation
+	# 11000b / CRRY	Quotient, non-zero rmd
DEC	# 24d / CRRY	Decimal result
7 **STWS** ↩ **CLR BIN** ↩ **CCR**		Start over in BIN with 7-bit word size
1101000 **ENTER** 1001 **ENTER**	2: # 1101000b 1: #1001b	Place lower and upper pieces in stack
110011 ↩ **DBL+**	11000b / CRRY	Same result as above

iii. Double Remainder

The **DBLR** function operates like **DBL÷** except that the remainder is returned instead of the quotient. However, unlike the double-divide function, if the quotient exceeds the current word size, an Overflow Error does not result. The remainder is determined similarly to the **RMD** function, with the sin of the result matching that of the dividend (numerator).

Example: Find the remainder when hexadecimal 329A4FCB12BFE5680 is divided by hexadecimal 71A484.

Initial conditions: Base: HEX Word size: 64
Complement Mode: Unsigned

HEX 64 **STWS** ↺ **UNSGN** ↻ **CLR** ↻ **CCR**

Keystrokes	Display Shows	Comments
29A4FCB12BFE5680 **ENTER 3 ENTER**	2: # 29A4FCB12BFE5680h 1: # 3h	Two-part dividend (66 bits long)
71A484 ↻ **DBLR**	# 6DD654h	Remainder

5. Real Numbers and Alpha Mode

A. Normal Real-Number Entry

i. Parser operation

The 16C Emulator allows real numbers (also called floating-point numbers in the HP16C manual) to be manipulated almost as easily as integers. While values entered without a decimal point or exponent of ten are automatically interpreted as integers, those numbers which DO contain a decimal point and/or exponent of ten will be interpreted to be reals. Consider the following data-entry examples:

(Initial conditions - Base: decimal, Word size: 32 bits,
Complement mode: 2's, Display mode: standard)

DEC 32 **STWS** **2's** **MODES STD** **CLR**

	Keystroke Sequence:	Command Line Shows:	Level-1 Value After ENTER is pressed:
a)	1 2	12	# 12d
b)	1 . 2	1.2	1.2
c)	1 2 .	12.	12
d)	1 . 2 E 2	1.2E2	120
e)	1 2 E 2	12E2	1200
f)	E 2	1E2	100

Note that if integral real numbers are desired, they must be entered with a terminating decimal point (as in case "c" above) or the parser will interpret the values as integers.

If the current base is octal or binary, the illegal numeric keys and the decimal point and **EEX** keys are normally disabled. However, turning on ALPHA mode re-enables the entire numeric keypad, keys A-F (on row 4) and the decimal point and **EEX** keys. When the above examples are run while the base is either octal or binary (with ALPHA-lock on), the same results will be generated. In addition, if digits are used which are illegal for the current base (such as 8 or 9 in octal or 2 through 9 in binary), the number will be interpreted to be a real regardless of whether the decimal point and/or **EEX** keys have been pressed. (See cases "j" and "k" below.) On the other hand if hex digits A through F are used in ALPHA mode with decimal, octal or binary base selected (as in case "l"), the result will be a syntax error when **ENTER** is pressed:

(Initial conditions - Base: octal, Word size: 32 bits,
Complement mode: 2's, Display mode: standard)

OCT α α 32. **STWS** 2's **MODES STD CLR**

	Keystroke Sequence:	Command Line Shows:	Level-1 Value After **ENTER** is pressed:
g)	1 2	12	# 12o
h)	1 α . 2	1.2	1.2
i)	1 α **EEX** 2	1E2	100
j)	1 α 8	18	18
k)	1 α . α 8	1.8	1.8
l)	1 α B 3	1B3	Invalid Syntax

Keystroke Sequence:	Command Line Shows:	Level-1 Value After **ENTER** is pressed:
m) **ATTN HEX ENTER**	1B3	# 1B3h
n) 1 . 2	1.2	1.2
o) 1 8	18	# 18h
p) 1 8 .	18.	18
q) 1 . 2 **EEX** 3	1.2E3	1200
r) 1 . **EEX** 3	1.E3	1000
s) 1 **EEX** 3	1E3	# 1E3h
t) 1 E 3	1E3	# 1E3h

Note that the behavior in case "s" above may seem strange at first, but is consistent with the fact that pressing the **EEX** key merely causes the letter "E" to be added to the command line. (This is true whether the emulator is activated or not.) The lesson to be learned here is to include a decimal point when entering integral real multiples of powers of ten when the current base is hex.

B. Real-Number Math

The 16C Emulator supports the four arithmetic functions plus change-sign at all times. In addition, fifteen scientific functions on HP48 keyboard row 4 may be accessed by toggling into the emulator's "**MTH**" mode. This operation,

activated by pressing the "**MTH**" key on row 2, causes the following changes to the emulator:

(1) The "16C" LCD annunciator changes to "**MTH**".

(2) Numeric digit keys 0 through 9 plus decimal point and **EEX** are legal while in ANY numeric base.

(3) Keyboard row-4 keys change from hex digits A to F plus the shifted bit-manipulation emulator functions to the original log, trig and exponential functionality of the regular HP48 for reals. Of the 18 functions assigned to these keys, only the blue-shifted functions on the first three keys (derivative, integral and summation) remain inactive.

(4) The square-root, x-squared and y-to-the-x functions also will accept integer arguments in emulator/**MTH** mode.

(5) The left- and right-shifted arithmetic keys provide the original HP48 object delimiter keys as labeled on the keyboard overlay to the right of the keys.

(6) Pressing the **MTH** key once more toggles out of emulator/**MTH** and back into emulator/16C mode.

Note that the 16C emulator keyboard overlay signifies the row-4 scientific functions by placing their identifying labels TO THE RIGHT of the assigned keys. The orange "-1" to the right of the first three keys in the row signifies the inverse trigonometric functions.

6. Format Conversions

A. Real-to-Binary and Binary-to-Real Conversions

Due to the fact that both integers and reals may be manipulated within the emulator environment, enhanced versions of the HP48's original "**B→R**" and "**R→B**" functions have been included in the environment. This allows the result of a calculation with either type of numeric values to be converted to the other type for further calculation. Of course like in the regular 48, when non-integral reals are converted, the numbers are rounded to the nearest integer. For the case when the current complement mode is unsigned, negative reals become zero after **R→B**. For signed modes, negative reals are rounded to the nearest negative integer. The examples below should further clarify this behavior:

(Initial conditions - Base: decimal, Word size: 32 bits,
Complement mode: Unsigned, Display mode: standard)

DEC 32 **STWS** ◀ **UNSGN** ◀ **MODES STD** ▶ **CLR**

Level-1 Value	Function	Resulting Value
3	**R→B**	# 3d
3.1	**R→B**	# 3d
3.8	**R→B**	# 4d
-3	**R→B**	# 0d / RNGE
(Switching to 1's complement mode: ◀ **1'S**)		
3	**R→B**	# 3d
3.1	**R→B**	# 3d
3.8	**R→B**	# 4d
-3	**R→B**	# -3d
-3.1	**R→B**	# -3d
-3.8	**R→B**	# -4d
0.9	**R→B**	# 1d
0.1	**R→B**	# 0d
-0.1	**R→B**	# 0d
-0.9	**R→B**	# -1d

i. Word Size Considerations

In the emulator (as in the HP48 itself), the current word size can affect the result of a real-to-binary conversion. If the number to be converted cannot be completely represented in the current word size, the lower portion of the number will be maintained in the stack as if it were intact. This also makes no special considerations for maintenance of the sign of the original or resulting number. If the new, smaller version of the number happens to have its most significant bit set, the emulator will interpret this as the presence of a negative number should the current complement mode be signed. Immediately increasing the word size at this point will reveal the upper hidden bits, however these are discarded as soon as any operation is performed with the smaller word size in force.

Also note that if the conversion to an integer would result in a value larger than can be represented in 64 bits, the resulting integer will have all bits set. This is interpreted by the emulator as a minus one (in decimal base) if the current complement mode is signed. Examples of this behavior are shown below:

(Initial conditions - Base: hex, Word size: 32 bits,
Complement mode: Unsigned, Display mode: standard)

HEX 32. **STWS** ◀ **UNSGN** ◀ **MODES STD**

Keystrokes	Resulting Value	Comments
256. **ENTER**	256	Entered value
R→B	# 100h	Converted to integer
B→R	256	Back to real: no change
8 **STWS**	256	Reduce word size
R→B	# 0h	Back to int: lower 8 bits
B→R	0	Back to real: remain zero

Keystrokes	Resulting Value	Comments
↩ **CLR** 32. **STWS** 511. **ENTER**	511	New starting value
R→B	# 1FFh	Converted to integer
DEC	# 511d	Switch to decimal base
B→R ↩ **2's** 8 **STWS**	511	To real and lower word size
R→B	# -1d	To int: a negative result
HEX	# FFh	Back to hex
32. **STWS**	# 1FFh	Restore word size: reveals hidden bits
CLR 25854.	25854	New starting value
R→B	# 64FEh	Converted to integer
DEC	# 25854d	Switch to decimal
B→R	25854	Back to real: original no.
8 **STWS R→B**	# -2d	Reduce word size and convert to int: negative result
HEX	# FEh	Switch to hex: lower 8 bits only
0 **+**	# FEh	Perform math on the value
B→R	-2	Back to real: hidden bits have been discarded

B. Converting Integers To and From Reals with FLOAT/FIXED

In the original HP16C calculator, the **FLOAT** function performs two functions:

(1) Turns on floating-point (real) decimal mode
(2) Converts the integers in the lowest two stack registers X and Y into the real number $y*2^x$

Since the 16C emulator permits entry of reals at any time, there is no need for the emulator's **FLOAT** function to change any modes. As a result, pressing **f FLT** converts stack levels one and two into the real "level-two times two to the level-one power". If the result is greater than $9.99999999999\times10^{499}$, the out-of- range flag is set. If no overflow is generated, the out-of-range flag is cleared.

Example: Convert hexadecimal values 4158E and 3F to a real number using **FLOAT**.

Initial conditions: Base: HEX Word size: 14 Complement Mode: 2's
Display mode: 4 FIX

HEX 32. **STWS f 2's f CLR g MODES 4 FIX**

Keystrokes	Display Shows	Comments
4158E **ENTER**	# 4158Eh	$4158E*2^{3F}$
3F **f FLT**	2.4687E24	

To reverse the operation of the **FLOAT** key in the HP16C calculator, one needs to press one of the integer base keys. This not only converts the stack level-one real to a pair of integers but turns on integer entry mode in the chosen base. Once again, since the 16C emulator can handle both integers and reals at the same time, there is no need for conversion from a real to a pair of integers to also change modes.

Considering the real number in stack level one to be in the form of Y times 2 to the X power, pressing **FXD** (**FIXED**) converts this real to the pair of integers X (in level one) and Y (in level two). If the current word size is less than 33 bits, executing the **FLOAT** function changes the word size to 33 bits. The word size is unaffected if it is 33 bits or greater. In order to achieve high precision, the mantissa value Y generated is a signed 32-bit integer. The decimal point is assumed to be to the right of the least significant bit in this mantissa value. As a result, the generated exponent of two (X) will be negative unless the original real number to be converted is greater than 2 to the 31st power (or greater than 2 billion).

Example:

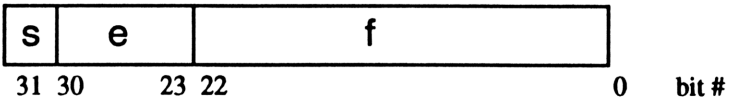
Initial conditions: Base: HEX Word size: 16 Complement Mode: 2's
Display mode: STD

HEX 16. **STWS** **2's** **CLR** **MODES STD**

Keystrokes	Display Shows	Comments
1.2E10 **ENTER**	12000000000	
	2: # B2D05E00h	Word size changed to
FXD	1: # 2h	33 bits
DEC	2: # 3000000000d	3 billion × 2 squared
	1: # 2d	

C. Converting To and From IEEE Floating-Point Format

In the HP16C calculator manual appendix are two programs for converting decimal real (HP16C floating-point mode) numbers to and from the proposed IEEE single-precision, floating-point binary format. These programs have been adapted and incorporated into the emulator library as the functions →**IEEE** and **IEEE**→. The IEEE proposed 32-bit standard is as follows:



where s = 1-bit sign
e = 8-bit biased exponent (ranging -127 to +128)
f = 23-bit fraction (mantissa)

A value v in the stack would be interpreted as follows:

Value of e	Value of f	Value of v
(a) 255	non zero	NaN (not a number)
(b) 255	zero	$\pm\infty$ (based on s)
(c) $0 < e < 255$	any	$-1^s * 2^{(e-127)} * 1.f$
(d) zero	non zero	$-1^s * 2^{-126} * 0.f$
(e) zero	zero	$-1^s * 0$

In the 16C emulator, the following conventions are used:

IEEE Number	Stack Value	Carry Flag	Out-of-Range Flag
0	0	clear	clear
-0	0	set	clear
±∞	±9.99999999999E499	set	set
Not a Number	$-1^s * 0.f * 2^{23}$	set	clear
Others	As defined above under (c) & (d)	set	clear

These two conversions turn on two's complement mode, base HEX and a word size of 32 bits.

Example:

Initial conditions: Base: HEX Word size: 32 Complement Mode: 2's
Leading Zeros Mode On, Display Mode: FIX 8

HEX 32. **STWS** **2's** **CLR** **MODES** **8** **FIX**
#000...

Keystrokes	Display Shows	Comments
80000000 **ENTER**	# 80000000h	-0
IEEE→	0.00000000 / CRRY	Carry set
CLR 7F800000	# 7F800000h	+∞
IEEE→	1.00000000E500 / CRRY, RNGE	Carry & Range Set
CLR 800000	# 00800000h	$2^{-126}*1.000...$
IEEE→	1.17549435E-38	
CLR 3F800001 **ENTER**	# 3F800001h	$2^{0}*(1.000....1)$
IEEE→	1.00000012	$=1+2^{-23}$
CLR 2.5E55 **ENTER**	2.500000E55	Value too large
→IEEE	# 7F800000h / RNGE	+∞
CLR 1.404E-45	1.40400000E-45	Smallest value
→IEEE	# 00000001h	

7. Using the VAR menu

While the 16C emulator is active, access to the HP48's **VAR** menu is permitted. There should be no difference in the behavior of user programs while the emulator is on, except that resulting integers in the stack will be displayed in the context of the active base, word size and complement mode. In addition, if a program halts to allow user input from the keyboard, the current entry modes of the emulator take precedence. For example, if a program halts from an **INPUT** instruction and expects real-number input, the user must make sure that reals get entered under the rules of the emulator. Either turning on **MTH** mode or including a decimal point or **EEX** will be necessary. Following pressing ↩ **CONT** the program should progress as expected.

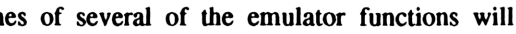
In the emulator, using the two shift keys in conjunction with the **VAR** keys works the same way as in the regular HP48: left-shift stores the stack level-one value into the object and right-shift recalls from the object into the stack.

A. Program HALT Cautions

Special precautionary measures must be observed with respect to encountering the **HALT** instruction in user programs running while the emulator is active. First and foremost, the **PRG CTRL** menu is not accessible, so single-stepping is not possible with the emulator on. In addition, the library is not able to accept the **KILL** instruction to terminate any halted conditions. If it is not acceptable to let the halted program run to completion via a **CONT** (continue) operation, the simplest way to release the halted environment would be to warm-start the calculator by pressing the **ON/C** key combination. This will clear the stack. (Check Appendix E for a method of preserving and restoring the stack if warm-start must be performed.)

8. Programming with Emulator Functions

Virtually every function in the 16C emulator can be used in a program object outside the emulator mode. The entire list of emulator functions can be scanned by turning on the emulator library menu and pressing **NXT** to see all the emulator function pages:

Keystrokes	Display Shows
↩ LIBRARY	
HP16C	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	
NXT	

The soft-key menu names of several of the emulator functions will differ outside the emulator environment than inside. While the emulator is on, those functions which are analogs of original HP48 functions have the same

names. However, since these functions behave differently than the regular functions, their names outside the emulator environment are changed so they can be differentiated. Most of these have been altered by having a decimal point added onto the end. In addition, some of the function names differ from the key labels on the emulator keyboard overlay. A list of these functions is shown below:

HP48 Function Name	16C Emulator Name Outside Emulator	16C Emulator Name Inside Emulator
+	ADDT	(on keyboard)
-	SUBT	(on keyboard)
*	x	(on keyboard)
/	÷	(on keyboard)
HEX	HEX.	HEX
DEC	DEC.	DEC
OCT	OCT.	OCT
BIN	BIN.	BIN
STWS	STWS.	STWS
SL	SL.	SL
SR	SR.	SR
RL	RL.	RL
RR	RR.	RR
ASR	ASR.	ASR
AND	AND.	AND
OR	OR.	OR
XOR	XOR.	XOR
NOT	NOT.	NOT
NEG	NEG.	NEG
ABS	ABS.	ABS
R→B	R→B.	R→B
B→R	B→R.	B→R
SQ	SQ.	(on keyboard)
^	^.	(on keyboard)
√	√.	(on keyboard)
(none)	ONES	1's
(none)	TWOS	2's
STWS	STWS.	STWS
RCWS	RCWS.	RCWS
(none)	no.B	#B

The above functions operate the same way outside the emulator environment as they do inside. The carry and out-of-range flags are set and/or cleared as usual, but the display annunciators are not on to visually reflect the changes.

Also, there are functions in the emulator library that do not get assigned to keys when the emulator is turned on. These are for use exclusively in program objects in order to better take advantage of the emulator functionality.

Below is a list of these functions:

SETC	**SETR**	**CRRY?**	**CMP?**
CLRC	**CLRR**	**RNGE?**	
<.	**>.**	**==.**	
≤.	**≥.**	**≠.**	

A. Test functions (CRRY?, RNGE?, CMP?, ==.,≠.,<., >., ≤., ≥.)

The test functions **CRRY?** and **RNGE?** allow user-generated program objects to test the status of the carry and out-of-range flags. These functions return a 1 to the stack if the flag is set and 0 if the flag is clear. Similarly, the "relational" tests for integers return 1 if the test is true and 0 if the test if false. Special considerations are built into these relational tests due to the differences between emulator integers and regular HP48 unsigned integers. For instance, in one's complement mode a positive and negative zero are considered to be numerically equal. Negative integers test to be less than positive ones in signed complement modes, despite the regular HP48's only being able to handle unsigned integers.

The **CMP?** test returns 0, 1 or 2 reflecting the current complement mode, be it unsigned, 1's or 2's. This may not only be used to determine the compliment mode, but may be used to determine whether the emulator simply is in a signed or unsigned mode. The sequence

```
IF CMP?
 THEN (code sequence for 1's and 2's comp.)
 ELSE (code sequence for unsigned mode)
 END
```

will execute the "**THEN**" code if **CMP?** returns a non-zero value or will execute the "**ELSE**" code if it returns zero.

B. Carry and Range functions SETC, CLRC, SETR, CLRR

These functions allow user programs to control the initial conditions of the carry and out-of-range flags. Due to the **TC**, **TR** and **CCR** functions existence in emulator mode, the set/clear carry/range functions are available only as library objects outside the emulator.

C. Writing Programs From Inside the Emulator Environment

It is indeed possible to construct user programs from inside the emulator environment. However, with limited access to HP48 menus, this may not be the best practice. For instance, without access to the **MATH PARTS**, **MATH PROB**, **PRG DSPL**, etc. menus, many of the HP48 functions are unavailable. On the other hand, if the menus provided are adequate, there should be no difficulties. In fact, this may be the easiest way to enter emulator functions into user programs, since they will be laid out on the entire keyboard (as opposed to in an eleven-page soft-key menu). From emulator MTH mode, one has access to the French quotes ("<<") in order to start the program object; then toggling out of MTH mode gives access to the normal 16C emulator keyboard. Note that since during editing of a program the PRG annunciator is on in the LCD, it will be impossible to visually determine whether emulator MTH mode is on or off. Also note that if the currently selected base happens to be one other than hexadecimal, several of the keypad keys (between 2 and F) may be disabled. Simply turn on ALPHA mode while entering the program object to get the full keypad back. Also remember that since in the emulator, the regular ALPHA keyboard is not available, naming the new user program object will be difficult if not impossible. Placing the object on the stack (via **ENTER**), exiting the

emulator and storing the object under the desired name is the best solution. The user program will then be accessible inside the emulator in the VAR menu.

D. Programming Examples

Below are a few programming examples which use the 16C emulator functionality in user program objects. Note that it is unnecessary to manually enter them into the calculator since these programs are available for downloading as a single directory ASCII file named "SAMPLES.DIR" on the floppy disk containing the emulator code. They are also stored as a directory called SAMPLES on the plug-in card. Go to the **LIBRARY** menu, then choose the key corresponding to the port in which the card is currently present. Press the key corresponding to the SAMP soft-key label and, if memory permits, the directory will be placed on the stack. Save it under a convenient name.

i. Recall and Set Emulator Status

Sometimes it is useful to be able to set the emulator base, word size and complement mode all in one step. Likewise, it may be worthwhile to recall this information and save it for future use. The following two short routines **RCST** (Recall Status) and **STST** (Store Status) perform this function:

```
RCST
<< BSS RCWS CMP? 3 →LIST >>
```

where **BSS** is the following:

```
BSS
<< { 10 8 2 16 } -11 FS? -12 FS? 2 × + 1 + GET >>
```

```
STST
 << OBJ→ DROP 1 + { UNSGN ONES TWOS } SWAP
     GET EVAL STWS
     CASE DUP 2 ==
     THEN BIN DROP
     END DUP 8 ==
     THEN OCT DROP
     END 10 ==
     THEN DEC
     END HEX
      END
 >>
```

RCST simply generates a list object containing the current base, word size and complement mode. **STST** takes such a list as an input and sets the base, word size and complement mode accordingly.

Example:

DEC 32 STWS ↵ TWOS

Function:	Display:
RCST	{ 10 32 2 }
{ 8 10 0 } **STST**	
RCST	{ 8 10 0 }
RCWS	10
CMP?	0
BSS	8

ii. Floating-point Base Calculator

Utilizing the functions of the emulator, it is possible to convert real numbers into floating-point numbers in the current integer base. The format would be:

±n.nnnnnnnnnnnZ ±mmm

where

±n.nnnnn	= Mantissa in the current base (up to 12 sig. digits)
Z	= Base letter indicator (H, E, C, B)
±mmm	= Exponent of the current base (expressed in decimal)

Just like HP48 reals may contain an "E" to indicate an exponent of ten, the above numbers would contain one of four possible base letters. The letter "C" was chosen for octal base since the letter "O" looks so much like a zero.

The main conversion routines are **R→FB** and **FB→R** (for "real" to and from "floating-point base") and work as follows:

Input value:	Keypress:	Resulting value:
1: ±x.xxxxxE±xxx	**R→FB**	1: "±n.nnnnnZ±mmm"
1: "±n.nnnnnZ±mmm"	**FB→R**	1: ±x.xxxxxE±xxx

The floating-point base number generated resides inside a string object, since it is not a valid HP48 object on its own. Since signed floating-point numbers are valid, the conversion from real to the floating-point base turns on two's complement mode if the current mode is unsigned. Also, in order to maintain maximum numerical accuracy, the word size is set to 64 bits. In order to specify the number of significant figures to the right of the radix point for floating-point base numbers, a routine called **FFIX** is used just like the HP48's **FIX** function.

Keystrokes	Display Shows	Comments
MODES STD HEX 5 **FFIX** 1.25 **ENTER**	1.25	Set to 5 significant figures
R→FB	"1.40000H0"	Convert to floating-pt. hex
FB→R	1.25	Returns to real
+/-	-1.25	Try a negative
R→FB	"-1.40000H0"	Negates result
CLR OCT 9.8 **EEX** 4	98000	Switch to octal
R→FB	"2.77320C5"	Floating-point octal
FB→R	98000	Return to real
10 **FFIX R→FB**	"2.7732000000C5"	Change to 10 figures and convert
3 **FFIX BIN FB→R**	98000	Change back to 3 figures and switch to binary
CLR MTH 1.74 **+/- EEX** 52 **+/- ENTER**	-1.74E-52	Try a negative with a negative exponent
R→FB	"-1.000B-172"	Floating-point binary result
FB→R	-1.67047794381E-52	Convert back: some accuracy loss

↩ CLR 11 FFIX		Try it with 11 places
1.74 +/- EEX 52 +/- ENTER	-1.74E-52	Same value
R→FB	"1.00001010101B-172"	Converted
FB→R	-1.73980930378E-52	Converted back: much higher accuracy

Now, it becomes fairly simple to create a floating-point base calculator which performs arithmetic on such numeric values embedded in strings. Routines **FADD, FSUB, FMULT** and **FDIV** perform addition, subtraction, multiplication and division between two floating-point base values in the stack:

Keystrokes	Display Shows	Comments
DEC HEX 5 **FFIX** ↩ **MODES STD**		Exit the emulator and set initial conditions
1.25 **R→FB**	"1.40000H0"	First value
3.18 **R→FB**	"3.2E148H0"	Second value
FADD	"4.6E148H0"	Add them
FB→R	4.43000030517	Convert back: small conversion error
5 ↩ **MODES FIX**	4.43000	Correct to 5 places
10 **FFIX STD** 1.25 **R→FB**	"1.4000000000H0"	Try 10 places
3.18 **R→FB FADD**	"4.6E147AE168H0"	Add second value
FB→R	4.43000000003	Correct to at least 10 places
"1.284A6H5" **ENTER**	"1.284A6H5"	Try adding hex and octal together
"4.45771C15" **FADD**	"9.2FE4012850H11"	Result is in current base
FB→R	1.61620694304E14	Convert back to real
DEC		Switch to decimal
R→FB	"1.61620694304E14"	Floating-point decimal version

The program listings are shown below:

```
FFIX
<< 11 MIN 'BSD' STO >>
```

BSD (The current number of desired significant figures - 0 to 11)

```
FADD
<< FB→R SWAP FB→R + R→FB >>

FSUB
<< FB→R SWAP FB→R SWAP - R→FB >>

FMULT
<< FB→R SWAP FB→R × R→FB >>

FDIV
<< FB→R SWAP FB→R SWAP / R→FB >>

R→FB
<< HDB SWAP DUP
   IF # 0h <.
   THEN 1 SF NEG.
   END
   IF DUP # 0h ==
   THEN DROP2 "0.00000000000" 1 BSD 2 + SUB LET + "0" +
   ELSE→STR DUP SIZE 4 - ROT + →STR SWAP DUP 3 3 SUB
     IF 1 FS?C
     THEN "-" SWAP +
     END "." + SWAP DUP SIZE 1 - 4 SWAP SUB + LET + SWAP +
   END
>>
```

```
HDB
<< 64 STWS
   IF CMP? NOT
   THEN TWOS
   END DUP
   IF 0 ==
   THEN DROP # 0h 0
   ELSE DUP ABS. DUP LOG BSD 1 + SWAP BSS LOG / - FLOOR
DUP NEG. SWAP BSS SWAP ^ ROT * .5 + IP ABS.
R→B. DUP
      IF BSS  R→B. 1 BSD
        START BSS  R→B. ×
        NEXT ==
      THEN BSS R→B. / SWAP 1 +
      ELSE SWAP
      END ROT
      IF 0 <
      THEN SWAP NEG. SWAP
      END
   END
>>

LET
<< { "E" "C" "B" "H" } -11 FS? -12 FS? 2 *+ 1 + GET >>

BSS
<< { 10 8 2 16 } -11 FS? -12 FS? 2 * + 1 + GET >>
```

```
FB→R
<< DUP 1 1 SUB
   IF "-" ==
   THEN 1 SF DUP SIZE 2 SWAP SUB
   END →STRING
   << 1 4
     FOR I STRING BASE I GET
       IF POS DUP
       THEN BSE I GET BNUM I GET ROT 4 'I' STO
       ELSE DROP
       END
     NEXT DUP STRING SWAP 1 - 1 SWAP SUB SWAP 1 +
     STRING DUP SIZE ROT SWAP SUB OBJ→ OVER
     SIZE 2 - - SWAP DUP 1 1 SUB SWAP DUP SIZE 3
     SWAP SUB + "#" SWAP + 4 ROLL + OBJ→ B→R. 3
     ROLLD ^ *
     IF 1 FS?C
     THEN NEG
     END
   >>
>>

BASE { "H" "E" "C" "B" }

BSE  { "h" "d" "o" "b" }

BNUM { 16 10 8 2 }
```

Appendix A. Classes of Operations

1.Active/Inactive Keys During Emulator Modes

The following is a list of active keys in each of the 16C Emulator modes:

Numeric Keypad While Current Base is BIN:

Digit keys 0 and 1, **+/-, ENTER, SPC**
(Digit keys 2 through F, Decimal Point, **EEX** are inactive)

Numeric Keypad While Current Base is OCT:

Digit keys 0 through 7, **+/-, ENTER, SPC**
(Digit keys 2 through F, Decimal Point, **EEX** are inactive)

Numeric Keypad While Current Base is DEC:

Digit keys 0 through 9, Decimal Point, **+/-, EEX, ENTER, SPC**
(Digit keys Athrough F, Decimal Point, **EEX** are inactive)

Numeric Keypad While Current Base is HEX:

Digit Keys 0 through F, Decimal Point, **+/-, EEX, ENTER, SPC**
(No keys in the keypad are inactive)

Keyboard Functions in "16C" Mode:

Numeric Keypad (as described above)
Base Keys **BIN, OCT, DEC, HEX**
MODES, PRG and **VAR** menu keys
MTH mode key, **CST** "main" menu page key
NXT, PREV menu page keys
UP, HOME, ' (tick), **STO, EVAL, REVIEW, SWAP,** arrow keys
SLn, SRn, FLT, FXD functions
Bit Manipulation Functions **SL, SR, RL, RR, RLn, RRn, RLC, RRC, RLCn,RRCn, LJ, ASR**

Leading-Zero's , **SET COMPL UNSGN, 1'S, 2'S**
EDIT, VISIT, DROP, CLR, DEL/"EXIT", back-arrow
ENTRY, α/"0...F", **CONT, OFF, ON/ATTN**
MSKL, MSKR, #B, SB, CB, B?
TC, TR, CCR
RAD, POLAR, LAST STACK, LAST ARG, LAST CMD, LAST MENU
→IEEE, IEEE→
Comma, carriage-return,π, ∠
+, -, ×, +, ABS, DBLR, RMD
Logic Functions **AND, OR, XOR, NOT**
B→R, R→B, DBL×, DBL+

Keyboard Functions in "MTH" Mode:

Numeric Keypad for DEC base
Base Keys **BIN, OCT, DEC, HEX**
MODES, PRG and **VAR** menu keys
MTH mode key, **CST** "main" menu page key
NXT, PREV menu page keys
UP, HOME, ' (tick), **STO, EVAL, REVIEW, SWAP,** arrow keys
SLn, SRn, FLT, FXD functions
SIN, ASIN, COS, ACOS, TAN, ATAN
$\sqrt{x}$, x^2, $\sqrt[x]{y}$, y^x, 10^x, **LOG**, $\frac{1}{x}$, e^x, **LN**
Leading-Zero's , **SET COMPL UNSGN, 1'S, 2'S**
EDIT, VISIT, DROP, CLR, DEL/"EXIT", back-arrow
ENTRY, α/"0...F", **CONT, OFF, ON/ATTN**
MSKL, MSKR, #B, SB, CB, B?
TC, TR, CCR
RAD, POLAR, LAST STACK, LAST ARG, LAST CMD, LAST MENU
→IEEE, IEEE→
Comma, carriage-return, π, ∠
+, -,×,+, ABS, DBLR, RMD
Object delimiters **( ), #, [ ], _, << >>, " ", { }, : :**

2.Flags Used by the Library

User Flags:

61:	Math Mode
62:	Leading-Zeroes Display Mode
63:	Carry Condition
64:	Out-of-Range Condition

System Flags:

-5 through -10 :	Current Word Size (like regular HP48 word size)	
-11, -12:	Current Base (like regular HP48)	
-13, -14:	Complement Mode:	
	Both clear:	Unsigned
	-13 Set / -14 Clear:	1's Complement
	-13 Set / -14 Set :	2's Complement

Appendix B. Function Summary:

Example:

ADDT	**(+)**
Function name in library	Function name in emulator

(Emu only) - Means function only appears in 16C Emulator Mode.
(Lib only) - Means function only appears in library menu outside emulator.
WS - Means "word size"

If a function name is listed only once, the representation in both the library directory and the emulator is the same.

Function	Stack Diagram	Explanation
α ("0...F") (Emu only)	→	Enables full numeric keypad keys 0 - F plus EEX and decimal point.
ABOUT (Lib only)	→	Displays info screen about the 16C Emulator Library
ABS. (ABS)	1:int → 1:int 1:real → 1:real	Takes the absolute value. If 2's complement mode, ABS(largest negative) causes out-of-range flag to be set; clears it otherwise.
ADDT (+)	2:int 1:int →1:int 2:int 1:real → 1:int 2:real 1:int → 1:int 2:real 1:real → 1:real Plus all other legal addition arguments	Addition. For the mixed cases, reals converted to integers via R→B before adding. Complement mode and word size respected. Carry and range flags modified as appropriate.

AND. (AND)	2:int 1:int → 1:int 2:real 1:real → 1:real	Combines integers logically bit-by-bit. Combines reals logically (values>0≡1) to produce 0 or 1.
ASR. (ASR)	1:int → 1:int	Shifts right while keeping sign bit intact. Sets carry if bit goes off end, clears it otherwise.
b (Emu only)	→	Right-shifted BIN on emulator MAIN soft-key menu page. Adds "b" to command line.
B?	2:int 1:int → 1:int 2:int 1:real → 1:int	Tests bit number n in integer in level 2 where n is in level 1 and returns 0 (if bit is clear) or 1 (if bit is set). Values between 0 and WS are valid. Reals in level 1 are rounded.
no.B (#B)	1:int → 1:int	Returns number of bits set in level 1 value.
B→R. (B→R)	1:int → 1:real	Converts integer to real.
BIN (Emu only)	→	Sets Binary base. Also enables keypad keys 0 and 1 only.
CB	2:int 1:int → 1:int 2:int 1:real → 1:int	Clear bit specified in level one. Values between 0 and WS are valid. Reals in level 1 are rounded.
CCR (Emu only)	→	Clears carry and range flags.
CLRC (Lib only)	→	Clears the carry flag
CLRR (Lib only)	→	Clears the out-of-range flag.

CMP:n (Emu only)	→	Three-way toggle between unsigned, one's and two's complement mode.
CMP? (Lib only)	→ 0, 1 or 2	Returns value of current complement mode
CRRY? (Lib only)	→ 0 or 1	Returns 1 if carry flag set, 0 otherwise
CST (Emu only) **("MAIN")**	→	Returns soft-key menu to 16C Emulator "main" menu.
d (Emu only)	→	Right-shifted DEC on emulator MAIN soft-key menu page. Adds "d" to command line.
DBLR	3:int 2:int 1:int → 1:int	Same as double divide but returns the remainder.
DBL×	2:int 1:int → 2:int 1:int	Multiplies two integers to produce a double-word result. Upper half is in level 1 with lower half in level 2. Clears range flag and does not affect carry flag.

DBL÷	3:int 2:int 1:int → 1:int	Divides double-word dividend in level 3 (lower half) and 2 (upper half) by single-word divisor in level 1 to produce single-word quotient. If result is larger than a single word, an overflow error results. Divide by zero causes an undefined result error. Clears range flag and affects carry flag (if result is not integral).
DEC (Emu only)	→	Sets Decimal base. Also enables numeric keypad keys 0 through 9, EEX and decimal point.
DEL (Emu only) **("EXIT")**	→	Exits 16C Emulator mode. Also performs DEL when editing objects.
EMUL (Lib only)	→	Enters 16C Emulator mode
FIXED (FXD)	1:real → 2:int 1:int	Returns 32-bit mantissa in level 2 and exponent of 2 in level 1 which is equivalent to real value input. If mode is unsigned, absolute value is taken on inputs and outputs.
FLOAT (FLT)	2:int 1:int → 1:real	Returns level-2 val × 2^(level-1 val). If result is too large, sets range flag.

h (Emu only)	→	Right-shifted HEX on emulator MAIN soft-key menu page. Adds "h" to command line.
HEX (Emu only)	→	Sets Hexadecimal base. Also enables numeric keypad keys 0 through F, EEX and decimal point.
IEEE→	1:int → 1:real	Converts IEEE 32-bit format to real. Sets word size = 32 bits.
→IEEE	1:real → 1:int	Converts real to IEEE 32-bit format. Sets word size = 32 bits.
LJ	1:int → 2:int 1:int	Left justify. Left justifies and returns value in level 2 and no. bits shifted left in level 1.
MLSHOW	→	Displays stack level-one object in 8-digit groups, occupying up to all 4 display lines for 64-bit binary integers.
MSKL	1:int → 1:int 1:real → 1:int	Create left-justified 1's binary mask with n bits. Values between 0 and ws are valid. If input is real, value is rounded first. Negative reals generate #0 mask. Negative integers have absolute value taken for mask. Values outside of WS cause Bad Argument Value error.

MSKR	1:int → 1:int 1:real → 1:int	Create right-justified 1's binary mask with n bits. Values between 0 and WS are valid. If input is real, value is rounded first. Negative reals generate #0 mask. Negative integers have absolute value taken for mask. Values outside of WS cause Bad Argument Value error.
MTH (Emu only) **("REALS")**	→	Turns on/off 16C Emulator/Math mode. Activates/deactivates numeric keypad keys 0 thru 9 plus EEX and decimal point. Also activates/deactivates most of HP48 original keyboard row-4 scientific functions plus 8 HP48 object delimiters.
NEG. (+/-)	1:int → 1:int 1:real → 1:real Plus all other legal negate arguments	Takes the negative of value in stack level one. Respects complement mode and word size. In unsigned mode, the negative of a nonzero integer causes the range flag to be set.
NOT. (NOT)	1:int → 1:int 1:real → 1:real	Complements all bits of integers. Converts real non-zeros to 0; 0 to 1.
o (Emu only)	→	Right-shifted OCT on emulator MAIN soft-key menu page. Adds "o" to command line.

OCT (Emu only)	→	Sets Octal base. Also enables numeric keypad keys 0 through 7 only.
ONES (1's)	→	Sets one's complement mode.
OR. (OR)	2:int 1:int → 1:int 2:real 1:real → 1:real	Combines integers logically bit-by-bit. Combines reals logically (values >0≡1) to produce 0 or 1.
R→B. (R→B)	1:real → 1:int	Converts real to integer. Rounds values first. Respects current complement mode and word size. Only shows WS bits of the converted value. Negative inputs become integer zero and sets range flag.
RCWS. (RCWS)	→ 1:real	Recall current word size.
RL. (RL)	1:int → 1:int	Rotate bits left. Sets carry if bit goes off end of word. Clears carry otherwise.
RLC	1:int → 1:int	Rotate 1 bit left thru carry. Sets carry if bit rotated off end, clears it otherwise.
RLCn	2:int 1:int → 1:int 2:int 1:real → 1:int	Rotate n bits left thru carry. Level 1 n value only valid when within ± WS. Level-2 value is rotated by rounded real if level 1 is real. Carry set if bit goes off end, cleared otherwise.

RLn	2:int 1:int → 1:int 2:int 1:real → 1:int	Rotate left by n bits. Level 1 value only valid when within ±WS. Negative level-1 value causes rotation in opposite direction. Level-2 value is rotated by rounded real if level 1 is real. Carry set if bit goes off end, cleared otherwise.
RMD	2:int 1:int → 1:int 2:int 2:real →1:int	Takes remainder after quotient of two input values. Real value is converted to integer (via **R→B.**) first.
RNG? (Lib only)	→1: 0 or 1	Returns 1 if range flag set, 0 otherwise
RR. (RR)	1:int → 1:int	Rotate bits right. Sets carry if bit goes off end of word. Clears carry otherwise.
RRC	1:int → 1:int	Rotate 1 bit right thru carry. Sets carry if bit rotated off end, clears it otherwise.
RRCn	2:int 1:int → 1:int 2:int 1:real → 1:int	Rotate n bits right thru carry. Level 1 only valid when within ±WS. Level-2 value is rotated by rounded real if level 1 is real. Carry set if bit goes off end, cleared otherwise.

RRn	2:int 1:int → 1:int 2:int 1:real → 1:int	Rotate right by n bits. Level 1 value only valid when within ±WS. Negative level-1 value causes rotation in opposite direction. Level-2 value is rotated by rounded real if level 1 is real. Carry set if bit goes off end, cleared otherwise.
SB	2:int 1:int → 1:int 2:int 1:real → 1:int	Set bit specified by stack level one. Values between 0 and WS are valid. Real level-1 values are rounded.
SETC (Lib only)	→	Sets the carry flag.
SETR (Lib only)	→	Sets the out-of-range flag.
SHOW BIN **SHOW DEC** **SHOW HEX** **SHOW OCT** (Emu only)	→	Left-shifted MAIN emulator soft keys. Briefly displays the integers in the LCD in HEX, DEC, OCT or BIN base.
SL. (SL)	1:int → 1:int	Shift left by 1 bit. Set carry if bit goes off end of word. Clear carry otherwise.
SLn	2:int 1:int → 1:int 2:int 1:real → 1:int	Shift left by n bits. Level 1 value only valid when within ±WS. Negative level-1 value causes rotation in opposite dir. Level-2 value is rotated by rounded real if level 1 is real. Carry set if bit goes off end, cleared otherwise.

SQ. (x^2)	1:int → 1:int 1:real → 1:real Plus any other legal HP48 input arguments	Squares level 1 value.
SR. (SR)	1:int → 1:int	Shift right by 1 bit. Sets carry if bit goes off end of word. Clears carry otherwise.
SRn	2:int 1:int → 1:int 2:int 1:real → 1:int	Shift right by n bits. Level 1 value only valid when within ±WS. Negative level-1 value causes rotation in opposite dir. Level-2 value is rotated by rounded real if level 1 is real. Carry set if bit goes off end, cleared otherwise.
STWS. (STWS)	1:real →	x<=0: WS=1 0<x<=64: WS=round(x) 64<x: WS=64
	1:int →	x<0: WS=|x| x=0: WS=64 0<x<64: WS=x 64<x: WS=64
SUBT (-)	2:int 1:int → 1:int 2:int 1:real → 1:int 2:real 1:int → 1:int 2:real 1:real → 1:real Plus all other legal subtraction arguments	Subtraction. For the mixed cases, reals converted to integers via **R→B.** before subtracting. Complement mode and word size respected. Carry and Range flags modified as appropriate.
TC (Emu only)	→	Toggles the carry flag.
TR (Emu only)	→	Toggles the out-of-range flag.
TWOS (2's)	→	Sets two's complement mode.

UNSGN	→	Sets unsigned integer mode.
XOR. (XOR)	2:int 1:int → 1:int 2:real 1:real → 1:real	Exclusive OR. Combines integers logically bit-by-bit. Combines reals logically (values >0≡1) to produce 0 or 1.
1/x (Emu only)	1:real → 1:real Plus any other legal HP48 input arguments	Reciprocal.
×	2:int 1:int → 1:int 2:int 1:real → 1:int 2:real 1:int → 1:int 2:real 1:real → 1:real Plus all other legal multiplication arguments	Multiplies the values. Does not affect carry flag but affects range flag. Reals converted via R →B. before multiplying.
÷	2:int 1:int → 1:int 2:int 1:real → 1:int Plus all other legal division arguments	Divides level 2 by level 1. Sets carry if result is not integral. Divide by zero causes Undefined Result error. Certain 2's complement mode situations causes range flag to be set.
#000... (Emu only) (**LZ** in **MODES** menu)	→	Toggles leading-zeros display mode.
√. ($\sqrt{x}$)	1:int → 1:int 1:real → 1:real Plus any other legal input arguments	Takes square root. If result of integer is non integral, carry flag is set.
^. (y^x)	2:int 1:int → 1:int 2:real 1:real → 1:real Plus any other legal HP48 input arguments	Level 2 to the level-1 power.

<. (Lib only)	2:int 1:int → 0 or 1 2:real 1:real → 0 or 1	Compares level 2 and level 1. If level 2 is less than level 1, returns 1; returns 0 otherwise. Respects complement modes.
>. (Lib only)	2:int 1:int → 0 or 1 2:real 1:real → 0 or 1	Compares level 2 and level 1. If level 2 is greater than level 1, returns 1; returns 0 otherwise. Respects complement modes.
≤. (Lib only)	2:int 1:int → 0 or 1 2:real 1:real → 0 or 1	Compares level 2 and level 1. If level 2 is less or equal to lvl 1, returns 1; returns 0 otherwise. Respects complement modes.
≥. (Lib only)	2:int 1:int → 0 or 1 2:real 1:real → 0 or 1	Compares level 2 and level 1. If level 2 is greater or equal to level 1, returns 1; returns 0 otherwise. Respects complement modes.
==. (Lib only)	2:int 1:int → 0 or 1 2:real 1:real → 0 or 1	Compares level 2 and level 1. If level 2 is equal to level 1, returns 1; returns 0 otherwise. Respects complement modes.
≠. (Lib only)	2:int 1:int → 0 or 1 2:real 1:real → 0 or 1	Compares level 2 and level 1. If level 2 is not equal to level 1, returns 1; returns 0 otherwise. Respects complement modes.

Appendix C. XLIB Numbers & Other Details

Function Name	XLIB # 364,	Legal Input Arguments: INTS	REALS	MIXED	Other	Affects: Crry	Rnge	Where in Library? Inside, Outside or Both (/alt. name)
EMUL	0	-	-	-	-	-	-	O
ABOUT	1	-	-	-	-	-	-	O
ADDT	2	Y	Y	Y	Y	Y	Y	B/ +
SUBT	3	Y	Y	Y	Y	Y	Y	B/ -
×	4	Y	Y	Y	Y	Y	Y	B
+	5	Y	Y	Y	Y	Y	N	B
SL.	6	Y	N	-	N	Y	N	B
SR.	7	Y	N	-	N	Y	N	B
RL.	8	Y	N	-	N	Y	N	B
RR.	9	Y	N	-	N	Y	N	B
RLn	10	Y	N	Y	N	Y	N	B
RRn	11	Y	N	Y	N	Y	N	B
LJ	12	Y	N	-	N	Y	N	B
ASR.	13	Y	N	-	N	Y	N	B
RLC	14	Y	N	-	N	Y	N	B
RRC	15	Y	N	-	N	Y	N	B
RLCn	16	Y	N	Y	N	Y	N	B
RRCn	17	Y	N	Y	N	Y	N	B
MSKL	18	Y	Y	-	N	N	N	B
MSKR	19	Y	Y	-	N	N	N	B
SB	20	Y	Y	Y	N	N	N	B
CB	21	Y	Y	Y	N	N	N	B
B?	22	Y	Y	Y	N	N	N	B
no.b	23	Y	N	-	N	N	N	B / #B
AND.	24	Y	Y	N	N	N	N	B
OR.	25	Y	Y	N	N	N	N	B
XOR.	26	Y	Y	N	N	N	N	B

Function Name	XLIB # 364,	Legal Input Arguments: INTS	REALS	MIXED	Other	Affects: Crry	Rnge	Where in Library? Inside, Outside or Both (/alt. name)
NOT.	27	Y	N	-	N	N	N	B
NEG.	28	Y	Y	-	Y	N	Y	B
ABS.	29	Y	Y	-	Y	N	Y	B
SLn	30	Y	N	Y	N	Y	N	B
SRn	31	Y	N	Y	N	Y	N	B
DBL×	32	Y	N	N	N	N	Y	B
DBL+	33	Y	N	N	N	N	Y	B
RMD	34	Y	N	N	N	N	N	B
DBLR	35	Y	N	N	N	N	N	B
→IEEE	36	N	Y	-	N	N	Y	B
IEEE→	37	Y	N	-	N	Y	Y	B
FLOAT	38	Y	N	N	N	N	Y	B
FIXED	39	N	Y	N	N	N	N	B
R→B.	40	N	Y	-	N	N	Y	B
B→R.	41	Y	N	-	N	N	N	B
==.	42	Y	Y	Y	Y	N	N	B
≠.	43	Y	Y	Y	Y	N	N	B
<.	44	Y	Y	N	N	N	N	B
>.	45	Y	Y	N	N	N	N	B
≤.	46	Y	Y	N	N	N	N	B
≥.	47	Y	Y	N	N	N	N	B
SETC	48	-	-	-	-	SET	N	O
CLRC	49	-	-	-	-	CLR	N	O
CRRY?	50	-	-	-	-	N	N	O
SETR	51	-	-	-	-	N	SET	O
CLRR	52	-	-	-	-	N	CLR	O
RNG?	53	-	-	-	-	N	N	O
UNSGN	54	-	-	-	-	N	N	B
ONES	55	-	-	-	-	N	N	B / 1's

Function Name	XLIB # 364,	Legal Input Arguments: INTS	REALS	MIXED	Other	Affects: Crry	Rnge	Where in Library? Inside, Outside or Both (/alt. name)
TWOS	56	-	-	-	-	N	N	B / 2's
CMP?	57	-	-	-	-	N	N	O
STWS.	58	Y	Y	-	N	N	N	B
RCWS.	59	-	-	-	-	N	N	O
√.	60	Y	Y	-	Y	Y	N	B / $\sqrt{x}$
SQ,	61	Y	Y	-	Y	N	N	B / x^2
^.	62	Y	Y	N	Y	N	Y	B / y^x
#000...	-	-	-	-	-	-	-	I
TC	-	-	-	-	-	Y	N	I
TR	-	-	-	-	-	N	Y	I
CCR	-	-	-	-	-	CLR	CLR	I
MTH	-	-	-	-	-	-	-	I

Appendix D. Error Messages

There are five basic error messages from the emulator:

Too Few Arguments	Not enough arguments for the operation, such as addition with only one stack value.
Bad Argument Type	Operation requires different argument(s) than are available in the stack, such as **B→R** with a real in stack level one.
Undefined Result	+, **DBL+**, **RMD** or **DBLR** with zero in the demoninator (including the case with zero in both numerator and denominator).
Overflow	Arguments in the stack will cause **DBL+** to return a result which is larger than the current word size.
Invalid Syntax	Evaluation of the command line containing an illegal expression.

Appendix E. When All Else Fails....

There is a possibility that at some point, an attempt may be made to:

(1) enter the 16C emulator
    (by pressing the EMUL key in the emulator menu); or
(2) exit the emulator
    (by pressing the DEL ("EXIT") key)

and nothing will happen except for the "wait" (hourglass) annunciator to be flashed. If this occurs, the remedy is to "warm-start" the HP48 by pressing the ON/C combination. Normal calculator behavior should resume at this point. Since the warm-start operation clears the stack, any important data should be saved first. If the emulator is active at the time, the stack can be saved and restored by the following keystroke sequence:

Keystrokes	Resulting Display	Comments
HEX PRG STK DEPTH	4: <stack item 3> 3: <stack item 2> 2: <stack item 1> 1: <depth>	Adds stack depth to stack level one
PRG OBJ →LIST	1: { <stack items>}	Entire stack in one list object
' ABCDEF STO		Using the HEX A thru F keys of row 4. Stack empties.
ON / C	[PARTS][PROB][.....	Machine blanks, then comes on with MTH menu active
VAR	[ABCDE] [.....	See new ABCDEF object in VAR menu

Keystrokes	Resulting Display	Comments
ABCDEF	1: { <stack items>}	List returned to stack level one
PRG OBJ OBJ→	4: <stack item 3> 3: <stack item 2> 2: <stack item 1> 1: <depth>	Unpack the list
DROP	3: <stack item 3> 2: <stack item 2> 1: <stack item 1>	Stack is restored

If any problems (or possible bugs) are encountered or you have suggestions for improvements, please feel free to contact Jake Schwartz at 135 Saxby Terrace, Cherry Hill, NJ 08003-4606 USA. Phone evenings and weekends is 609-751-1310.

Index

HP16C Emulator Library for the HP48S/SX