41CL Memory Functions

Every effort has been made to ensure the accuracy of the information contained herein. If you find errors or inconsistencies please bring them to our attention.

Notice:

Acknowledgements:

Ángel Martin provided the CPYBNK function, as well as valuable feedback during the development of this software.

Table of Contents

Introduction

The *41CL Memory Functions* provide extra features and functions for directly accessing the hardware (physical memory, registers or peripheral registers) of the 41CL. If you rarely deal with the hardware resources of the 41CL these functions will probably not be useful to you. The *41CL Memory Functions* are independent of the normal *41CL Extra Functions* and *41CL Extreme Functions*, and can be plugged into the 41CL on an as-needed basis using the "YFNF" mnemonic to program the MMU.

This manual covers version -3A. Earlier versions of the *41CL Memory Functions* had a different set of functions and are no longer supported.

The *41CL Memory Functions* are grouped into six categories:

1. The Address Pointer functions implement four address pointers, which allow easy access to locations in memory without having to continually specify an address.

2. The General Memory functions include word-oriented PEEK and POKE functions that use the address pointers, as well as functions to compare the contents of two regions of physical memory, or find specific data in a region of physical memory.

3. The Page Status functions provide a way to keep track of which pages of physical memory have been used, in either Flash or RAM.

4. The CL Expanded Memory functions provide a way to keep up to three alternate copies (numbered 1, 2 and 3) of the regular 41C memory (number 0), either as backups or to change the "personality" of your machine.

5. Most of the miscellaneous functions provide housekeeping for the remainder of the *41CL Memory Functions*. One miscellaneous function provides a way to copy programs for the secondary banks of physical modules, something that the regular *41CL Extra Functions* and *41CL Extreme Functions* are not capable of doing.

6. The 41C Register functions allow easy access to the data registers and peripheral registers of the 41C.

7. The CL Configuration functions allow you to initialize the MMU registers in memory with a standard set of alternate configurations.

Some of the *41CL Memory Functions* use dynamic paging, where code is transiently loaded to Page 4 while the functions are executed. Any image loaded to Page 4 (like *Library-4*) will be temporarily displaced by these functions and then restored before the function finishes. **No physical module that uses Page 4 should be present in the calculator when using the *41CL Memory Functions*. The HP-IL Module with the Printer Function Switch in the "DISABLE" position uses Page 4.**

Address Pointer Functions

Four address pointers are implemented to allow easy access to locations in 41CL memory. These pointers are called the *X Address Pointer* (XP), the *Y Address Pointer* (YP), the *Z Address Pointer* (ZP), and the *T Address Pointer* (TP). There are functions to store, recall, increment, swap, subtract, roll up and roll down the pointers (similar to the normal 41C stack.)

Address pointers can hold any value between 0x000000 and 0xFFFFFF, but all of the functions that use address pointers to access a physical memory location automatically verify that the physical memory address is valid. The ranges of acceptable physical memory addresses are shown in the table below:

	Lowest address	Highest address
V2 Flash memory	0x000000	0x0FFFFF
V3/V4 Flash memory	0x000000	0x1FFFFF
V5 Flash memory	0x000000	0x3FFFFF
V2 RAM memory	0x800000	0x83FFFF
V3/V4/V5 RAM memory	0x800000	0x87FFFF
41C Register memory	0x800000	0x803FFF

The majority of functions in the *41CL Memory Functions* only work with physical addresses, but since not all physical memory addresses are valid, one invalid address range has been reserved to serve as an alias for 41C peripheral registers. This address range is shown in the table below.

	Lowest address	Highest address
41C Peripheral Registers	0xFFF000	0xFFFFFF

The underlined digits in the table above contain the actual address portion of the pointer. This address alias is only allowed with the various 41C Register Functions and the functions that increment or add to pointers. **Incrementing or adding to an aliased pointer only affects the least-significant digit, because each peripheral address consists of a two-digit peripheral identifier and a one-digit register address.**

Each of the address pointers are stored in two memory locations, as shown below. Even though the address pointers are stored in separate memory locations, all pointer operations with a normal pointer use the entire six digits of a pointer.

	Memory address	Data			
		4	3	2	1
X Address Pointer	0x804020	0	X2	X1	X0
	0x804021	0	X5	X4	X3
Y Address Pointer	0x804022	0	Y2	Y1	Y0
	0x804023	0	Y5	Y4	Y3
Z Address Pointer	0x804024	0	Z2	Z1	Z0
	0x804025	0	Z5	Z4	Z3
T Address Pointer	0x804026	0	T2	T1	T0
	0x804027	0	T5	T4	T3

STOXP
STOYP
STOZP
STOTP (6-digit hex address in ALPHA register)

Executing **STOXP** (*Store X Address Pointer*) writes directly to the *X Address Pointer* at addresses 0x804020 and 0x804021.

Executing **STOYP** (*Store Y Address Pointer*) writes directly to the *Y Address Pointer* at addresses 0x804022 and 0x804023.

Executing **STOZP** (*Store Z Address Pointer*) writes directly to the *Z Address Pointer* at addresses 0x804024 and 0x804025.

Executing **STOTP** (*Store T Address Pointer*) writes directly to the *T Address Pointer* at addresses 0x804026 and 0x804027.

No range check is performed when a pointer is stored. Instead, any function that uses a pointer automatically performs the appropriate range check before the function executes.

The data for these four commands is a normal six-digit hexadecimal number, as shown below.

	ALPHA register					
	6	5	4	3	2	1
Address Pointer value	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***

RCLXP
RCLYP
RCLZP
RCLTP

Executing **RCLXP** (*Recall X Address Pointer*) reads directly from the *X Address Pointer* at addresses 0x804020 and 0x804021.

Executing **RCLYP** (*Recall Y Address Pointer*) reads directly from the *Y Address Pointer* at addresses 0x804022 and 0x804023.

Executing **RCLZP** (*Recall Z Address Pointer*) reads directly from the *Z Address Pointer* at addresses 0x804024 and 0x804025.

Executing **RCLTP** (*Recall T Address Pointer*) reads directly from the *T Address Pointer* at addresses 0x804026 and 0x804027.

These four functions return with the contents of the address pointer in the display (Run mode only) and the ALPHA register, formatted as shown below.

	Display and ALPHA register					
	6	5	4	3	2	1
Address Pointer value	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***

XP+
YP+
ZP+
TP+

Executing **XP+** (*Increment X Address Pointer*) increments the *X Address Pointer* by one.

Executing **YP+** (*Increment Y Address Pointer*) increments the *Y Address Pointer* by one.

Executing **ZP+** (*Increment Z Address Pointer*) increments the *Z Address Pointer* by one.

Executing **TP+** (*Increment T Address Pointer*) increments the *T Address Pointer* by one.

All four of the pointer increment functions return with the new contents of the Address

pointer in both the display (Run mode only) and the ALPHA register, formatted as shown below.

	Display and ALPHA register					
	6	5	4	3	2	1
Address Pointer value	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***

XP+A

Executing **XP+A** (*Add ALPHA Register to X Address Pointer*) adds the hexadecimal number in the ALPHA register to the *X Address Pointer*. The hex number in the ALPHA register is treated as a 16's-complement value. That is, the number must be between +8,388,607 and -8,388,608. This function allows the *X Address Pointer* to step forward or backward through addresses in increments other than one.

The data for this command is a normal six-digit hex number, as shown below. Leading zeros do not need to be present, so the number can be scaled as appropriate for an aliased address.

	ALPHA register					
	6	5	4	3	2	1
hex number	***D5***	***D4***	***D3***	***D2***	***D1***	***D0***

This function returns with the new contents of the *X Address Pointer* in both the display (Run mode only) and the ALPHA register, formatted as shown below.

	Display and ALPHA register					
	6	5	4	3	2	1
X Address Pointer value	***X5***	***X4***	***X3***	***X2***	***X1***	***X0***

XP+X

Executing **XP+X** (*Add X Register to X Address Pointer*) adds the number in the X register to the *X Address Pointer*. The number in the X register must be between -999 and 999. This function allows the *X Address Pointer* to step forward or backward through

addresses in increments other than one.

This function returns with the new contents of the *X Address Pointer* in both the display (Run mode only) and the ALPHA register, formatted as shown below.

Display and ALPHA register					
6	5	4	3	2	1
X Address Pointer value	***X5***	***X4***	***X3***	***X2***	***X1***	***X0***

XP<>YP

Executing **XP<>YP** (*Exchange X Address Pointer with Y Address Pointer*) exchanges the *X Address Pointer* with the *Y Address Pointer*.

This function returns with the new contents of the *X Address Pointer* in both the display (Run mode only) and the ALPHA register, formatted as shown below.

Display and ALPHA register					
6	5	4	3	2	1
X Address Pointer value	***X5***	***X4***	***X3***	***X2***	***X1***	***X0***

YP-XP

Executing **YP-XP** (*Subtract X Address Pointer from Y Address Pointer*) subtracts the *X Address Pointer* from the *Y Address Pointer*, **but does not affect either pointer**. Both pointers are treated as unsigned numbers, but the result is a normal 6-digit 16's-complement number. Aliased pointers are not allowed.

This function returns with the result of the subtraction in both the display (Run mode only) and the ALPHA register, formatted as shown below.

Display and ALPHA register					
6	5	4	3	2	1
difference between pointers	***D5***	***D4***	***D3***	***D2***	***D1***	***D0***

RUPP

Executing **RUPP** (*Roll Up Address Pointers*) replaces the *T Address Pointer* with the *Z Address Pointer*, the *Z Address Pointer* with the *Y Address Pointer*, the *Y Address Pointer* with the *X Address Pointer*, and the *X Address Pointer* with the *T Address Pointer*.

This function returns with the new contents of the *X Address Pointer* in both the display (Run mode only) and the ALPHA register, formatted as shown below.

	Display and ALPHA register					
	6	5	4	3	2	1
X Address Pointer value	***X5***	***X4***	***X3***	***X2***	***X1***	***X0***

RDNP

Executing **RDNP** (*Roll Down Address Pointers*) replaces the *X Address Pointer* with the *Y Address Pointer*, the *Y Address Pointer* with the *Z Address Pointer*, the *Z Address Pointer* with the *T Address Pointer*, and the *T Address Pointer* with the *X Address Pointer*.

This function returns with the new contents of the *X Address Pointer* in both the display (Run mode only) and the ALPHA register, formatted as shown below.

	Display and ALPHA register					
	6	5	4	3	2	1
X Address Pointer value	***X5***	***X4***	***X3***	***X2***	***X1***	***X0***

General Memory Functions

The General Memory functions provide tools that make it easier to work directly with the physical memory of the 41CL.

The new **YPEEK+** and **YPOKE+** functions use the *X Address Pointer* to hold the address information, and automatically increment this pointer by one after each peek or poke operation. This simplifies writing contiguous data to memory because you don't have to specify the address every time. Unlike the regular **YPOKE** function, the **YPOKE+** function works with both Flash memory and RAM, so be very careful to make sure that the *X Address Pointer* contains the correct address.

The **YDIFF?** function allows you to compare the data in two regions of memory, while the **YSRCH?** function allows you to search memory for a specific word of data. Both of these functions use the *Memory Mask Register* to select which bits will contribute to the compare or search.

YDIFF?

Executing **YDIFF?** (*Search for Differences*) compares the word at the address specified by the *X Address Pointer* with the word at the address specified by the *Y Address Pointer*, using the *Memory Mask Register* contents to select which bits to compare. Both pointers are then incremented and the compare continues, until cancelled by a keyboard input. This function is not programmable.

For each iteration, the *X Address Pointer* is displayed to show the progress of the search. If the two words match, both pointers are incremented and the process repeats. If the two words do not match both the *X Address Pointer* and the corresponding data value are written to both the ALPHA register and the display and the function pauses to wait for a keyboard input before continuing.

The display is formatted as shown below in the case of a difference:

	Display and ALPHA Register										
	11	10	9	8	7	6	5	4	3	2	1
Physical address and data	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***D3***	***D2***	***D1***	***D0***

The compares are stopped and restarted using the **R/S** key. Compares continue until cancelled using the backspace key.

When this function is cancelled the last difference is displayed and is present in the ALPHA register. If no difference has been found the last address compared, which is the current *X Address Pointer*, is displayed but the ALPHA register is unaffected.

YSRCH? (4-digit hex word in ALPHA register)

Executing **YSRCH?** (*Search for Data*) compares the word at the address specified by the *X Address Pointer* with the word in the ALPHA register, using the *Memory Mask Register* contents to select which bits to compare. The pointer is then incremented and the compare continues until cancelled by a keyboard input. This function is not programmable.

The data to compare with memory is taken from the right-most four digits of the ALPHA register. All other information in the ALPHA register is ignored.

	ALPHA Register			
	4	3	2	1
Compare data	***D3***	***D2***	***D1***	***D0***

For each iteration, the *X Address Pointer* is displayed to show the progress of the search. If the two words do not match, the pointer is incremented and the process repeats. If the two words match both the *X Address Pointer* and the corresponding data value are written to the ALPHA register and the display and the function pauses to wait for a keyboard input before continuing.

The display is formatted as shown below in the case of a match:

	Display and ALPHA Register										
	11	10	9	8	7	6	5	4	3	2	1
Physical address and data	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***D3***	***D2***	***D1***	***D0***

The search can be stopped and restarted using the **R/S** key. Searches continue until cancelled using the backspace key.

When this function is cancelled the last match is displayed and is present in the ALPHA register. If no difference has been found the search data is displayed and the ALPHA register is unaffected.

YPEEK+

Executing **YPEEK+** (*Read Word Using Pointer and Increment Pointer*) reads directly from either Flash or RAM memory. This function uses the *X Address Pointer* to hold the memory address information, and automatically increments this pointer after the read.

The *X Address Pointer*, prior to the increment, and the read data are returned in both the display (Run mode only) and the ALPHA register in the format shown below. This is the same format used by the normal **YPEEK** function.

	Display and ALPHA Register										
	11	10	9	8	7	6	5	4	3	2	1
Physical address and data	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***D3***	***D2***	***D1***	***D0***

YPOKE+ (4-digit hex write data in ALPHA register)

Executing **YPOKE+** (*Write Word Using Pointer & Increment Pointer*) writes directly to either Flash or RAM memory. This function uses the *X Address Pointer* to hold the memory address information, and automatically increments this pointer after the write.

The data to write to memory is taken from the right-most four digits of the ALPHA register. All other information in the ALPHA register is ignored. This allows this function to use the same format as the normal **YPOKE** function.

	ALPHA Register			
	4	3	2	1
Write data	***D3***	***D2***	***D1***	***D0***

In Run mode, before writing to Flash this function requires user confirmation, by sending ***FL WR OK?*** to the display. Pressing the **R/S** key confirms that the write should be performed. Pressing any other key will cancel the function. If no key is pressed within 10 seconds the function will be cancelled and a ***NULLED*** error message will be written to the display.

When writing to Flash memory the **YPOKE+** function does not check for pre-existing data in the Flash location, so be careful, and remember that writing Flash memory can only change a "1" to a "0" and never vice-versa. This function does not allow writing to the Operating System area (pages 0x000 to 0x007) of Flash memory.

Page Status Functions

The Page Status functions allow you to keep track of which pages of Flash or RAM have been used. These functions cannot tell you what the pages are being used for; only whether or not they might be empty.

These functions take advantage of the fact the memories on in the 41CL are sixteen bits wide, while HP-41 instructions are only ten bits wide. Two of the otherwise unused bits are used by the NEWT microprocessor for instruction-by-instruction control of the Turbo mode, but the other four bits are not normally used, especially by the first word of a page. The figure below shows how the bits are organized in a 41CL memory word.

nibble 3				nibble 2				nibble 1				nibble 0			
3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
Status		Turbo		unused		10-bit HP-41 instruction									

The two status bits in the upper nibble are used to indicate an unused page according to the table below.

Status		Meaning
0	0	Programmed Flash or RAM page
0	1	Reserved
1	0	Unprogrammed RAM page
1	1	Unprogrammed Flash page

Since an unprogrammed word of Flash contains 0xFFFF, and in the 41CL the Flash is only written by page, if the two status bits are “11” the Flash page has almost certainly not been programmed. If necessary, the **YCRC** function can be used to verify a completely unprogrammed page of Flash memory. An unprogrammed page of Flash memory will have a CRC of 0x53D36BD2.

Unfortunately, RAM can power up with any data so there is no automatic way to determine that a page of RAM has not been used. The **MEMINI** function can be used to initialize the first word of selected RAM pages with “10” in the status bits. Then if the first word of a RAM page is ever written with normal HP-41 instructions or data, these bits will automatically be changed to “00,” giving an indication that the page has been used. This indication is not foolproof, because individual words of RAM can be written at any time.

PMEMCHK (page address in ALPHA register)

Executing **PMEMCHK** (*Programmable Memory Status Check*) reads the contents of the first word in the selected page and checks the status bits to see if the page may have been used.

The figure below shows the formatting required for the address in the ALPHA register for the **PMEMCHK** function.

	ALPHA register		
	3	2	1
single address	***P5***	***P4***	***P3***

When executed from the keyboard this function returns the page number and ***USED*** in the display if the status bits indicate that the page has been used, and the page number and ***UNUSED*** in the display if the status bits indicate that the page has not been used.

When **PMEMCHK** is used in a program, if the status bits indicate that the page has been used the next program line will be executed; if the status bits indicate that the page has not been used the next line in the program is skipped.

MEMCHK (first and last page address in ALPHA register)

Executing **MEMCHK** (*Memory Status Check*) reads the contents of the first word in each selected page and checks the status bits to see if the page may have been used. A range of pages is specified, and either Flash or RAM addresses are allowed. This function is not programmable.

The figure below shows the formatting required for the address pair in the ALPHA register for the **MEMCHK** function. This function also accepts the same single-address format used by the **PMEMCHK** function.

	ALPHA register						
	7	6	5	4	3	2	1
		first				last	
address range	***P5***	***P4***	***P3***	**>**	***P5***	***P4***	***P3***

When executed, this function returns the page number and ***USED*** in the display if the status bits indicate that the page has been used, and the page number and ***UNUSED*** in

the display if the status bits indicate that the page has not been used. If an address range is specified the checks continue until halted with the **R/S** key. The checks can be stopped and restarted using the **R/S** key. The function is cancelled using the backspace key.

PMEMINI (page address in ALPHA register)

Executing **PMEMINI** (*Programmable Initialize Memory Status*) writes bits 15 and 14 of the first word the selected page to indicate that the page is unused. Because of the way that Flash memory works, only RAM addresses are allowed. Note that the first five pages of RAM (addresses 0x800 to 0x804) are used by the 41CL, and this function will not allow those pages to be marked as unused.

The figure below shows the formatting required for the address in the ALPHA register for the **PMEMINI** function.

	ALPHA register		
	3	2	1
single address	***P5***	***P4***	***P3***

MEMINI (first and last page address in ALPHA register)

Executing **MEMINI** (*Initialize Memory Status*) writes bits 15 and 14 of the first word of each selected page to indicate that the page is unused. Either a single page or range of pages may be specified, but only RAM addresses are allowed. This function is not programmable. Note that the first five pages of RAM (addresses 0x800 to 0x804) are used by the 41CL, and this function will not allow those pages to be marked as unused.

The figure below shows the formatting required for the address pair in the ALPHA register for the **MEMINI** function. This function also accepts the same single-address format used by the **PMEMINI** function.

	ALPHA register						
	7	6	5	4	3	2	1
		first				last	
address range	***P5***	***P4***	***P3***	**-**	***P5***	***P4***	***P3***

CL Alternate Memory Functions

The microprocessor used in the 41C and the NEWT microprocessor used in the 41CL are capable of addressing 4096 registers of data memory, each consisting of seven bytes. The 41CL reserves four 4096-word pages of RAM to implement this register address space. However, the 41C Operating System is only capable of accessing one fourth of the microprocessor register address space, or one 4096-word page of RAM. This leaves three pages of RAM unused, and the 41CL Memory Functions can use these RAM locations to store alternate versions of the 41C register memory.

When using these *Alternate Memory* versions of the 41C register memory it is important to remember that the register addresses used by the microprocessor are different from the user-visible register addresses of the calculator, and the 41C Operating System normally manages this memory resource. As a consequence, you should always be careful when using these alternate versions of the 41C register memory. More information about how the 41C Operating System uses its data memory can be found in the 41CL Memory Reference, as well as a number of historical books and documents.

The functions here divide each of the three 4096-word pages of *Alternate Memory* into sixteen 256-word sections, as shown below. Each of these sections can be individually selected to be read, written or exchanged with the original 41C register memory. For convenience, these functions also can use three mnemonic identifiers for the different regions of this memory. This is also shown in the table.

Register address	41C use	Physical address	Selector	Identifier
000 - 03F	OS use	0x800000 - 0x8000FF	0	S
040 - 07F	X Memory	0x800100 - 0x8001FF	1	X
080 - 0BF		0x800200 - 0x8002FF	2	
0C0 - 0FF	Main Memory	0x800300 - 0x8003FF	3	M
100 - 13F		0x800400 - 0x8004FF	4	
140 - 17F		0x800500 - 0x8005FF	5	
180 - 1BF		0x800600 - 0x8006FF	6	
1C0 - 1FF		0x800700 - 0x8007FF	7	
200 - 23F	X Memory	0x800800 - 0x8008FF	8	X
240 - 27F		0x800900 - 0x8009FF	9	
280 - 2BF		0x800A00 - 0x800AFF	10	
2C0 - 2FF		0x800B00 - 0x800BFF	11	
300 - 33F	X Memory	0x800C00 - 0x800CFF	12	
340 - 37F		0x800D00 - 0x800DFF	13	
380 - 3BF		0x800E00 - 0x800EFF	14	
3C0 - 3FF		0x800F00 - 0x800FFF	15	

The first section, with register addresses 0x000-0x03F, is used exclusively by the 41C Operating System. This section is where the user-visible stack and ALPHA registers reside, along with the flags and other information. In reality, only the first sixteen register addresses in this section are visible to the Operating System, and the remainder is never used. Be very careful when restoring or exchanging this section, because one of the locations contains a bit pattern that the Operating System uses to determine whether or not to signal the ***MEMORY LOST*** condition.

The two sections with register addresses 0x040-0x0BF are the memory normally found in the *HP Extended Functions/Memory* module.

The five sections with register addresses 0x0C0-0x1FF are the Main Memory for the 41C. This is where key assignment buffers, alarm buffers, I/O buffers and User programs are stored.

The four sections with register addresses 0x200-0x2FF are the memory found in the first Extended Memory Module, while the four sections with register addresses 0x300-0x3FF are the memory found in the second Extended Memory Module.

The easiest way to use these function to read, write and exchange with *Alternate Memory* is useing the identifiers shown in the table. The ***S*** identifier selects the Status area of the 41C memory, while the ***M*** identifier selects the Main Memory area of the 41C memory, and an ***X*** identifier selects the Extended Memory area of the 41C memory.

Any one, two, or three identifiers, in any order, can be used with the functions here to select the section(s) on which to operate. The identifiers are taken from the three least-significant digits of the ALPHA register.

	ALPHA register		
	3	2	1
Identifiers	***I3***	***I2***	***I1***

The selection information for these commands can also be specified explicitly with a four-digit hex number. Each of the sixteen bits in this hex number corresponds to a section, as was also shown in the table above. This hex number is taken from the ALPHA register.

	ALPHA Register			
	4	3	2	1
Block selector value	***D3***	***D2***	***D1***	***D0***

This means that an **S** identifier is the same as 0x0001 when using a hex number, an **M** identifier is the same as 0x00F8 when using a hex number, and an **X** identifier is the same as 0xFF06 when using a hex number.

Only the values 1, 2 and 3 are valid for the *Alternate Memory* selection for these functions.

PSTOM (alternate memory in X register, identifier in ALPHA register)

Executing **PSTOM** (*Programmable Store to Alternate Memory Area*) copies data from the default memory area to the corrresponding locations in the *Alternate Memory Area* as selected by the contents of the X Register.

STOM (prompts for alternate memory, identifier in ALPHA register)

Executing **STOM** (*Store to Alternate Memory Area*) copies data from the default memory area to the corrresponding locations in the *Alternate Memory Area* as selected by the prompt. This function is not programmable.

PRCLM (alternate memory in X register, identifier in ALPHA register)

Executing **PRCLM** (*Programmable Recall from Alternate Memory Area*) copies data from the *Alternate Memory Area* as selected by the contents of the X Register to the corresponding locations in the default memory area.

RCLM (prompts for alternate memory, identifier in ALPHA register)

Executing **RCLM** (*Recall from Alternate Memory Area*) copies data from the *Alternate Memory Area* as selected by the prompt to the corresponding locations in the default memory area. This function is not programmable.

PEXM (alternate memory in X register, identifier in ALPHA register)

Executing **PEXM** (*Programmable Exchange with Alternate Memory Area*) exchanges data from the *Alternate Memory Area* as selected by the contents of the X Register with the corresponding locations in the default memory area.

EXM (prompts for alternate memory, identifier in ALPHA register)

Executing **EXM** (*Exchange with Alternate Memory Area*) exchanges data from the *Alternate Memory Area* as selected by the prompt with the corresponding locations in the default memory area. This function is not programmable.

Miscellaneous Functions

Some of the General Memory functions require data masking information to control which bits of the memory data are to be used by the functions. The *Memory Mask Register* is used to store this information. Functions are provided to store, recall, and preconfigure this mask information.

A number of functions in the *41CL Memory Functions*, the *41CL Extra Functions* and *41CL Extreme Functions* use page addresses or page address pairs in the ALPHA register. Two functions are provided to increment these types of addresses to reduce keystrokes and program sizes.

One miscellaneous function is included that allows copying code from bank-switched modules. Normally only the first bank is visible to code not running in the image, but the **CPYBNK** function overcomes this limitation.

STDMSK

Executing **STDMSK** (*Standard Data Mask*) writes 0x03FF directly to the *Memory Mask Register* at address 0x804026. This is the correct mask value to use when searching or comparing 10-bit HP-41 instructions. The *Memory Mask Register* is not automatically initialized.

STOMSK (4-digit hex mask value in ALPHA register)

Executing **STOMSK** (*Store Data Mask*) writes directly to the *Memory Mask Register* at address 0x804026. Only *Memory Mask Register* bit positions containing a one will contribute to a search or compare. So, to search/compare the entire 16 bits, use a mask value of 0xFFFF.

The data for this function is a normal four-digit hex number, as shown below. If the ALPHA register contains more than four digits, only the four rightmost digits will be used.

ALPHA Register			
4	3	2	1
Memory Mask value	***M3***	***M2***	***M1***	***M0***

RCLMSK

Executing **RCLMSK** (*Recall Data Mask*) reads directly from the *Memory Mask Register* at address 0x804026.

This function returns with the mask value in both the display (Run mode only) and the ALPHA register, formatted as shown below.

	Display and ALPHA Register			
	4	3	2	1
Memory Mask value	***M3***	***M2***	***M1***	***M0***

DST+ (page address in ALPHA register)

Executing **DST+** (*Increment Destination Address*) increments the page address in the ALPHA register by one, moving to the next page. This is useful when using functions that require a page address, particularly in programs. This function can also be used to increment a 41C register address in the ALPHA register.

The figure below shows the formatting required for the address in the ALPHA register for the **DST+** function. In Run mode the new address is displayed in the same format.

	ALPHA register		
	3	2	1
single address	***P5***	***P4***	***P3***

SRCDST+ (source and destination page address in ALPHA register)

Executing **SRCDST+** (*Increment Source/Destination Address Pair*) increments both addresses in a typical source/destination address pair in the ALPHA register by one, moving to the next pages. This is useful when using functions that require a source and destination address pair, particularly in programs.

The figure below shows the formatting required for the addresses in the ALPHA register

for the **SRCDST+** function. In Run mode the new address is displayed in the same format.

ALPHA register						
7	6	5	4	3	2	1
	source			destination		

source and destination addresses **P5 P4 P3 > P5 P4 P3**

RCLBP
RCLGP
RCLPP

Executing **RCLBP** (*Recall Memory Buffer Pointer*) loads the contents of the *Memory Buffer Pointer* to the *X Address Pointer*.

Executing **RCLGP** (*Recall GET Buffer Pointer*) loads the contents of the *GET Buffer Pointer* to the *X Address Pointer*.

Executing **RCLPP** (*Recall PUT Buffer Pointer*) loads the contents of the *PUT Buffer Pointer* to the *X Address Pointer*.

The *Memory Buffer Pointer*, *GET Buffer Pointer* and *PUT Buffer Pointer* always point at the *Memory Buffer Page* at page address 0x805.

STOBP
STOGP
STOPP

Executing **STOBP** (*Store Memory Buffer Pointer*) loads the contents of the *X Address Buffer Pointer* to the *Memory Buffer Pointer*.

Executing **STOGP** (*Store GET Buffer Pointer*) loads the contents of the *X Address Buffer Pointer* to the *GET Buffer Pointer*.

Executing **STOPP** (*Store PUT Buffer Pointer*) loads the contents of the *X Address Buffer Pointer* to the *PUT Buffer Pointer*.

The *Memory Buffer Pointer*, *GET Buffer Pointer* and *PUT Buffer Pointer* always point at the *Memory Buffer Page* at page address 0x805, even though the upper three nibbles of

the *X Address Pointer* are stored by these commands.

CPYBNK (prompts for bank, plus source and destination pages)

Executing **CPYBNK** (*Copy Bank*) copies the code from any bank of one page to the main bank in another page. The destination page should be RAM for the copy to occur. This function is most useful when copying banked code from a physical module into 41CL memory, because the normal **YMCPY** function is only able to copy from bank 1 of a physical module.

41C Register Functions

The 41C Register functions allow direct access to the entire 41C register address space, including the registers that are not normally accessable by the 41C Operating System. Data registers can be accessed using the physical address of the register. Peripheral registers can only be access using the alias capability of the register pointers.

Each 41C data register consists of seven bytes of information, and in the 41CL the entire 41C register memory is mapped to dedicated pages in physical memory, with each register occupying four words of RAM. A 41C data register address is twelve bits (three hex digits) in length, for a total of 4096 registers.

Two functions translate between the normal three-digit hex 41C data register address and the corresponding physical memory address in the *X Address Pointer*. The *X Address Pointer* (and the *Y Address Pointer*, if necessary) are used to access the register(s) in physical memory. Remember that the physical 41C register address is different from the user-visible register addresses of the calculator.

Peripheral registers are also seven bytes, but are addressed slightly differently from data registers. While peripheral registers are still three digits in length, normally a 41C peripheral will decode the upper two digits as a "peripheral identifier" and then use the least-significant digit to select a register within the peripheral.

Those 41C Register functions that require data expect a fourteen-digit hex number in the ALPHA register, as shown below. Because the display can only show twelve digits, the two most-significant digits will scroll out of the display to the left. Leading zeros do not need to be present, and if the ALPHA register is empty, a value of all zeros will automatically be used by the functions.

ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
D13	***D12***	***D11***	***D10***	***D9***	***D8***	***D7***	***D6***	***D5***	***D4***	***D3***	***D2***	***D1***	***D0***

REG2XP (3-digit hex register address in ALPHA register)

Executing **REG2XP** (*Load Register Address to X Address Pointer*) converts a three-digit hexadecimal 41C data register address to the actual physical address of that register and loads the result to the *X Address Pointer*.

The 41C register address for this function is a normal three-digit hex number, as shown

below. If the ALPHA register contains more than three digits, only the three right-most digits will be used.

ALPHA Register		
3	2	1
Physical 41C register address	***R2***	***R1***	***R0***

XP2REG

Executing **XP2REG** (*Translate X Address Pointer to Data Register Address*) converts the physical address in the *X Address Pointer* to a 41C data register address.

This function returns with the 41C register address in both the display (Run mode only) and the ALPHA register, formatted as shown below.

Display and ALPHA Register		
3	2	1
Physical 41C register address	***R2***	***R1***	***R0***

YPEEKR+

Executing **YPEEKR+** (*Read Register Using Pointer and Increment Pointer*) reads directly from the 41C data register area of RAM memory or the 41C peripheral register. This function uses the *X Address Pointer* to hold the register address information, and automatically increments this pointer to the next register address after the read.

The register data is returned in both the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

YPOKER+ (14-digit hex value in ALPHA register)

Executing **YPOKER+** (*Write Register Using Pointer and Increment Pointer*) writes directly to the 41C data register area of RAM memory or to the 41C peripheral register. This function uses the *X Address Pointer* to hold the register address information, and automatically increments this pointer to the next register address after the write.

The data written to the register is returned in the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

YPEEKR

Executing **YPEEKR** (*Read 41C Register Using Pointer*) reads directly from the 41C data register area of RAM memory or the 41C peripheral register. This function uses the *X Address Pointer* to hold the register address information. The *X Address Pointer* is not affected by this function.

The register data is returned in both the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

YPOKER (14-digit hex value in ALPHA register)

Executing **YPOKER** (*Write 41C Register Using Pointer*) writes directly to the 41C data register area of RAM memory or to the 41C peripheral register. This function uses the *X Address Pointer* to hold the register address information. The *X Address Pointer* is not affected by this function.

The data written to the register is returned in the display (Run mode only) and the

ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

SETREG (14-digit hex value in ALPHA register)

Executing **SETREG** (*Set bits in Register Using Pointer*) writes the 41C data or peripheral register addressed by the *X Address Pointer* with the logical-OR of the register data and the data in the ALPHA register. In other words, for each bit position in the ALPHA register that is set to one, the corresponding bit in the 41C register will be set to one. All other 41C register bits are not affected.

The data written to the register is returned in the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

CLRREG (14-digit hex value in ALPHA register)

Executing **CLRREG** (*Clear bits in 41C Register Using Pointer*) writes the 41C data or peripheral register addressed by the *X Address Pointer* with the logical-AND of the register data and the inverse of the data in the ALPHA register. In other words, for each bit position in the ALPHA register that is set to one, the corresponding bit in the 41C register will be set to zero. All other 41C register bits are not affected.

The data written to the register is returned in the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

COMREG (14-digit hex value in ALPHA register)

Executing **COMREG** (*Complement bits in Register Using Pointer*) writes the 41C data or peripheral register addressed by the *X Address Pointer* with the logical-XOR of the register data and the data in the ALPHA register. In other words, for each bit position in the ALPHA register that is set to one, the corresponding bit in the 41C register will be complemented. All other 41C register bits are not affected.

The data written to the register is returned in the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

LDREG

Executing **LDREG** (*Load Register Using Pointers*) writes the 41C data register addressed by the *Y Address Pointer* with the data from the 41C data register addressed by the *X Address Pointer*. The register addressed by the *X Address Pointer* is unaffected. 41C peripheral registers are not allowed for this function.

The data written to the register is returned in the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

EXREG

Executing **EXREG** (*Exchange Registers Using Pointers*) exchanges the data in the 41C data register addressed by the *Y Address Pointer* with the data from the 41C data register addressed by the *X Address Pointer*. 41C peripheral registers are not allowed for this function.

The data written to the register addressed by the *X Address Pointer* is returned in the display (Run mode only) and the ALPHA register in the format shown below.

Display and ALPHA Register													
14	13	12	11	10	9	8	7	6	5	4	3	2	1
R13	***R12***	***R11***	***R10***	***R9***	***R8***	***R7***	***R6***	***R5***	***R4***	***R3***	***R2***	***R1***	***R0***

Configuration Functions

The 41CL supports fifteen alternate configurations, where each configuration is a set of MMU programming that selects the images plugged into pages 4 through F. Populating these fifteen alternate configurations can be a time-consuming process, so the the **CFGINI** function initializes twelve of the alternate configurations with a set of images that you may find useful, either directly or as a starting point for your own custom configurations.

In addition, aliases for these different configurations have been implemented in the *Image Database* (IMDB) to make it easier to remember which configuration is which. Of course you are still free to customize your 41CL as you like.

CFGINI

Executing **CFGINI** (*Initialize Alternate Configurations*) loads twelve of the alternate configurations according to the table below.

The table below shows how alternate configurations 4 through F are initialized by this function. Alternate configurations 1 through 3 are not affected.

	Configuration											
PLUG	9PWR	9MTH	9SCI	9INF	9PRG	9ELE	9MEC	9MAP	9PLY	9BGM	9CST	9HIL
Page	4	5	6	7	8	9	10	11	12	13	14	15
4	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB	4LIB
5												
6	OSX3	OSX3	OSX3	OSX3		OSX3	OSX3	OSX3	OSX3	OSX3		
7	YFNX	YFNX	YFNX	YFNX	YFNX	YFNX	YFNX	YFNX	YFNX	YFNX	YFNX	
8	PWRX	SM44	SERI	16CS	YUPS	ETS5	ETS3	METX	FUNS	CHES	YUPS	OSX3
9	WARP		SLVF	CRTO	16CS			5MAD			YFNF	DEV2
A	XPMM	4MTI	FRML	RCSN		EEFD	ETS4	WORD		PPOK		
B	4TBX		XTAT	PWRX						MAZZ		EXTI
C	ROMX	Z4DL	CURV	BASI		UNIT	UNIT	CITY	RUBK	RGME		EXIO
D	HEP2					CIRC	MCHN	CLND	4WIN			YFNX
E		ADVG	EPTC	RNDZ		EENG	MENG	SUD1	AGAM	GSWP		DACQ
F		ZMAT	ELIX	NONL		ETS9	NBOD	CRTO				

Almost all of these alternate configurations assume that the *41CL Extreme Functions* will be loaded in page 7, but starting with version -4D of the *41CL Library Functions*, the *41CL Extreme Functions* do not need to be in the same pages between primary and alternate configurations. This greatly simplifies switching configurations.

If you do not want any of the pages of an alternate configuration loaded when switching configurations, mark the appropriate pages as Locked in the MMU to prevent them from being modified. For example, if you don't normally use OSX3 or want to use a printer, just Lock page 6 to prevent it from being loaded from an alternate configuration. The same will be true for page 4 and an HP-IL module with the printer disabled.

These alternate configurations are just a starting point, and given the wide variety of images available in the 41CL, you may find yourself customizing them to suit your own individual tastes or needs. The "9xxx" aliases used with the **PLUG** function in the *41CL Extreme Functions* work independent of the contents of the alternate configurations. Alternate configurations 1-3 are also assigned aliases: "9CFA" for alternate configuration 1, "9CFB" for alternate configuration 2, and "9BAS" for alternate configuration 3.

The simplest way to customize an alternate configuration is to load the default contents, modify everything to suit your needs, and then write it back to the desired alternate configuration. The only caveat is that you must be careful with the *41CL Extreme Functions* (YFNX), because you cannot move this image while executing functions contained within the image. Moving YFNX requires disabling the MMU to perform the move.

It is also possible to customize the alternate configurations by writing directly to the corresponding MMU registers in RAM. This is significantly more complicated than the method described above, and is not recommended.

The MMU registers are not affected by the ***MEMORY LOST*** condition, but may be lost when the batteries are removed from the 41CL. If you have customized your alternate configurations you may want to create a backup copy by copying the MMU page in RAM (page 0x804) to an unused page in Flash memory for permanent storage.

Error Messages

The table below list all possible error messages returned by the *41CL Memory Functions*, along with the meaning of the message.

Error Message	Function	Meaning
ADDR ERROR	**MEMCHK** **MEMINI** **PMEMCHK** **PMEMINI** **YDIFF?** **YPEEK+** **YPOKE+** **YSRCH?**	Address is outside of valid Flash or RAM address range
	XP2REG **YPEEKR+** **YPOKER+** **SETREG** **CLRREG** **COMREG** **LDREG** **EXREG**	Address is not in 41C register range (0x800000- 0x803FFF), is an unaligned to a 41C register address, or is an improper aliased address
DATA ERROR	All functions using hex	Invalid hexadecimal digit
	PEXM **PRCLM** **PSTOM**	Invalid *Alternate Memory* selection
DST=ROM	**PMEMINI** **MEMINI**	Destination address in Flash
FMT ERROR	**MEMCHK** **MEMINI** **SRCDST+**	Format error in address pair
NO BANK	**CPYBNK**	Bank does not exist
NONEXISTENT	**XP+X**	Number in X register out of range
OS AREA	**PMEMINI** **MEMINI**	Attempted operation on Operating System area of RAM
	YPOKE+	Attempted operation on Operating System area of Flash
SRC=ROM	**PMEMINI** **MEMINI**	Source address in Flash
SSS>DDD	**MEMCHK** **MEMINI**	Start address greater than Finish address

Revision History

12/26/2018	Version -3A original issue.
08/14/2019	Typo in table on P. 32
08/26/2019	Elaborated the CFGINI function operation.
12/06/2019	Modify format for double-sided printing.
12/23/2020	Add sentence about mnemonic for loading the MMU.
03/01/2022	Adjust configurations for XROM conflicts