41CL Extreme 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:

Table of Contents

Introduction

The *41CL Extreme Functions* provide extra functionality beyond the normal *41CL Extra Functions*. If you rarely plug in new software images the normal *41CL Extra Functions* are probably sufficient for your needs. But if you frequently change the MMU configuration of your 41CL, the *41CL Extreme Functions* will make using your 41CL much easier.

The *41CL Extreme Functions* support four different sets of MMU configurations, allowing you to completely change the personality of your 41CL with just one command. MMU entries now support a "Locked" status, which prevents you from accidently writing over critical MMU programming (like the location of the *41CL Extreme Functions*.) Other MMU functions allow searching the MMU programming for current contents, either by mnemonic or address.

The latest *Image Database* contains additional information about images, grouping them into searchable groups of similar functionality and providing size and placement restriction information for the **PLUG** functions.

Many of the *41CL Extreme Functions* prompt for user input instead of requiring that the information be entered into the ALPHA register first, for a more user-friendly feel.

You should be familiar with the operation of the normal *41CL Extra Functions* before attempting to load or use the new *41CL Extreme Functions*. By default, the normal *41CL Extra Functions* are always available to you, so you will need to use one of those functions to insert the *41CL Extreme Functions* into the calculator. For example, the sequence **ALPHA YFNX ALPHA XEQ ALPHA PLUG1L ALPHA** plugs the *41CL Extreme Functions* into Page 8.

Once the *41CL Extreme Functions* are active, every time the calculator is turned on, and at the start of a number of commands, the MMU programming is checked for hardware conflicts. If a physical module is detected plugged into a port, the MMU entry for the corresponding page is automatically marked as Locked and MMU translation is disabled for that page. This check is done for pages 6 through F, so if you plug in a physical module that uses Page 4 (like a Service Module, or an HP-IL module with the printer disabled) you will need to manually disable MMU translation for that page.

This check for hardware conflicts is only one way, so it will be necessary to manually remove the Locked status if you remove a physical module.

After the check for hardware conflicts, the *41CL Extreme Functions* are automatically marked with the Locked status to protect you from accidentally removing these functions. If you really want to disable the *41CL Extreme Functions*, you'll need to disable the MMU (automatically restoring the *41CL Extra Functions*) and then use one of the **UPLUG** functions to disable the *41CL Extreme Functions*.

The three sets of secondary MMU registers are not automatically initialized, so always use the **MMUCLS** (*Clear Secondary MMU Registers*) function to initialize all of the secondary MMU registers before you start using them. You can also use the **MMUCLP** (*Clear Primary MMU Registers*) function to initialize all of the unlocked primary MMU registers, but be aware that because Pages 0-3 are not affected by the **MMUCLR** (*Clear MMU Registers*) function in the *41CL Extra Functions*, they may power up marked as Locked and may need to be manually unlocked.

The *41CL Extreme 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 the *41CL Extreme 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 Extreme Functions*. Note that the HP-IL Module with the Printer Function Switch in the "DISABLE" position uses Page 4.**

The dynamic paging used in the *41CL Extreme Functions* **requires** that the *41CL Library Functions* be present in Flash memory in the page immediately following the *41CL Extreme Functions*. The page address for this library is 0x00B.

Unlike the identically named functions in *41CL Extra Functions*, the **YFERASE** (*Erase Flash Memory Sector*) and the **YFWR** (*Write Flash Memory Page*) functions in the *41CL Extreme Functions* are not required to run out of RAM. However, if either of these functions are operating on the Flash memory sector that contains them, the *41CL Extreme Functions* must be resident in RAM.

MMU Functions

The Memory Management Unit (MMU) in the 41CL contains four sets of registers: a primary set (number 0) and three secondary sets (numbers 1, 2 and 3.) The primary set of registers are used whenever the MMU is globally enabled, to map the addresses used by a program to the addresses used by the physical memory devices. The three sets of secondary MMU registers exist only for pages 4 through F, and allow you to store alternate "personalities" in the calculator for easy access.

Global MMU Functions

MMUCLP

Executing **MMUCLP** (*Clear Primary MMU Registers*) clears all of the primary MMU registers that are not marked as Locked. The secondary MMU registers (sets 1, 2 and 3) are not affected. The *41CL Extreme Functions* are automatically marked as Locked, so this function will never remove the MMU programming for these functions.

This function is different from the normal **MMUCLR** function in that it operates on all sixteen pages (instead of only pages 4-F) and it does not clear the MMU programming for Locked pages.

MMUCLS

Executing **MMUCLS** (*Clear Secondary MMU Registers*) clears all three sets of secondary MMU registers. The primary MMU registers are not affected. This function ignores any Locked status present in the secondary MMU registers.

MMUDIS

Executing **MMUDIS** (*Disable MMU*) clears the global MMU enable bit in the hardware. This automatically reassigns the normal *41CL Extra Functions* to Page 7, but does not affect the MMU register contents. **MMUDIS** results in all pages (except for the Operating System, Extended Functions, Time Functions and *41CL Extra Functions*) being fetched from the Ports rather than from internal memory. Remember that an HP-IL module should not be inserted into the calculator while the MMU is disabled.

MMUEN

Executing **MMUEN** (*Enable MMU*) while the *41CL Extreme Functions* are active checks pages 6 through F for hardware conflicts, where a physical module is detected in a page and the MMU register is enabled for that page. If a physical module is detected plugged into a port, the MMU entry for the corresponding page is automatically marked as Locked and MMU translation is disabled for that page.

MMU?

Executing **MMU?** (*Test MMU Enable*) tests the state of the global MMU enable bit, returning with ***NO*** in the display if the MMU is disabled and ***YES*** in the display if the MMU is enabled. When used in a program, if the MMU is enabled the next program line will be executed; if the MMU is disabled the next line in the program is skipped.

Locked Status Functions

The MMU allows individual pages to be marked as Locked. When a page is Locked the MMU registers for that page are unaffected by functions that would otherwise update the MMU registers for that page. Only the **MMUCLR** (in the default *41CL Extra Functions*) and the **UNLOCK** functions will affect an MMU register that is marked as Locked.

A page can be marked as Locked even though nothing has been virtually plugged into the page. This feature can be used to protect physical modules from conflicting with a virtual image in the same page.

Modifying the MMU contents for a Locked page requires a two-step process. First, the Locked status must be removed using the **UNLOCK** function, and then the MMU can be reprogrammed for that page.

When any version of the *41CL Extreme Functions* are inserted into a page using the **PLUG** or **PPLUG** functions the page is automatically marked as Locked. This protects the *41CL Extreme Functions* from accidently being removed or overwritten.

LOCK	**(prompts for Page) OR (Page in X register)**

Executing **LOCK** (*Lock Page*) sets the Locked status for the selected Page in the primary set of MMU registers. This function prompts for the Page number in Run mode, and takes the Page number from the X register in Program mode.

UNLOCK	**(prompts for Page) OR (Page in X register)**

Executing **UNLOCK** (*Unlock Page*) removes the Locked status for the selected page, which then allows the MMU registers for the page to be updated. This function prompts for the Page number in Run mode, and takes the Page number from the X register in Program mode.

Note that it is not possible to remove the Locked status from the page containing the 41CL Exteme Functions.

LOCK?	**(prompts for Page) OR (Page in X register)**

Executing **LOCK?** (*Test Locked Status*) tests the Locked status of the selected Page in the primary set of MMU registers. This function prompts for the Page number in Run mode, and takes the Page number from the X register in Program mode.

In Run mode the function returns with ***NO*** in the display if the Locked status for the page is not set and ***YES*** in the display if the Locked status is set. When used in a program, if the Locked status is set the next program line will be executed; if Locked status is not set the next line in the program is skipped.

MMU Register Set Functions

There are four sets of MMU registers, called set 0 through set 3. The active, or primary, MMU registers are set 0, while sets 1, 2 and 3 are secondary. There are primary MMU registers for all sixteen pages, but the secondary sets of MMU registers exist only for pages 4 through F.

EXCFG (prompts for MMU Register Set)

Executing **EXCFG** (*Exchange MMU Configuration*) exchanges the contents of the primary set of MMU registers (set 0) with the contents of the selected set of MMU registers, for pages 4 through F. This function is not programmable.

Any MMU registers in the primary set (set 0) that are marked as Locked will not be exchanged and will be listed in the ALPHA register and the display by the function. Thus, executing **EXCFG** with MMU register set 0 selected will report all of the pages marked with the Locked status, without making any changes to the MMU programming.

RCLCFG (prompts for MMU Register Set)

Executing **RCLCFG** (*Recall MMU Configuration*) loads the contents of the selected set of MMU registers into the primary set (set 0) of MMU registers, for pages 4 through F. This function is not programmable.

Any MMU registers in the primary set (set 0) that are marked as Locked will not be written, and will be listed in the ALPHA register and the display by the function. Thus, executing **RCLCFG** with MMU register set 0 selected will report all of the pages marked with the Locked status, without making any changes to the MMU programming.

STOCFG (prompts for MMU Register Set)

Executing **STOCFG** (*Store MMU Configuration*) stores the contents of the primary set of MMU registers into the selected set of MMU registers, for pages 4 through F. This function is not programmable.

Special MMU Functions

The Special MMU Functions allow you to enable and disable MMU translation for the Operating System pages (0-3 and 5). Normally these pages are never mapped by the MMU, for obvious reasons.

Unlike the normal MMU enable bit, the enable bit for the special MMU operation is not preserved when the calculator is turned off. This provides a fail-safe way to restore the native Operating System.

MAPDIS

Executing **MAPDIS** (*Disable Special MMU Mapping*) clears the special MMU enable bit, which automatically restores the native Operating System. Since this function will normally be executed from a modified Operating System, the function automatically returns using the normal 41C function call/return convention.

MAPEN

Executing **MAPEN** (*Enable Special MMU Mapping*) sets the special MMU enable bit inside the NEWT microprocessor. This function automatically enables mapping of the Operating System pages, but only if the MMU is globally enabled.

Turbo Functions

The Turbo Functions give you control over the speed of the calculator. The performance in Turbo mode is not linear, because some operations must always occur at normal speed. For example, scanning the keyboard (which is done once per program line) always executes at normal speed. Similarly, accessing the display for any reason always executes at normal speed. All accesses of a physical Port occur at normal speed, along with a number of timing loops in the Operating System and Timer functions.

Turbo modes increase the current consumption during normal operation, but have no effect during idle time (between keypress) or during the time when the calculator is off. Like the Turbo mode performance, the current consumption as a result of Turbo mode is not linear.

The Turbo mode is preserved during idle time, as well as when the calculator is turned off.

PTURBO (Turbo value in X register)

Executing **PTURBO** (*Programmable Turbo Mode Select*) transfers the contents of the X register to the Turbo mode controller. Only the values 0, 1 (both of which disable Turbo mode), 2, 5, 10, 20 and 50 are valid.

TURBO (prompts for two-digit value)

Executing **TURBO** (*Turbo Mode Select*) prompts the user for a two-digit Turbo mode value. Only the values 00, 01 (both of which disable Turbo mode), 02, 05, 10, 20 and 50 are valid. This function is not programmable.

TURBO?

Executing **TURBO?** (*Test Turbo Mode*) queries the state of the Turbo mode controller, and the current Turbo speed is returned in the X register. The results returned will be one of ***0***, ***2***, ***5***, ***10***, ***20*** or ***50***. The stack is lifted before the result is written to the X register if Stack Lift is enabled.

Page Management Functions

The Page Management Functions allow you to virtually plug and unplug module images from the calculator without having to remember specific memory addresses. These functions use the Image Database in Flash memory to determine the location and size of the module image, which provides an easy way for you to control the configuration of the calculator. Refer to the Image Identifier Table section for the mnemonics for the different module images.

You should avoid XROM conflicts when using module functions in programs. Refer to the original HP documentation for details. XROM Conflicts can be circumvented by copying a module image to RAM and then manually modifying the XROM number before plugging this modified image into a Port. Use the **CHKXROM** function to check for XROM conflicts.

You must also avoid hardware conflicts with physical modules. When a module image is virtually plugged into a Port no physical module that uses that Port address can be plugged into the calculator. The only exceptions are those modules or peripherals that use dedicated addressing, shown in the table below. However, even these modules and peripherals must avoid an addressing conflict. The 82182A Time Module can be plugged into any Port without conflict.

41C Module or Peripheral	Page Address
82104A Card Reader	E
82143A Printer	6
82160A HP-IL Module	4 or 6, and 7
82242A IR Printer Module	6

Some third-party modules can be addressed independently of their physical location, but you must still avoid address conflicts with these modules.

PPLUG **(image identifier and start page in ALPHA register) OR (image identifier in ALPHA register, start page in X register)**

Executing **PPLUG** (*Programmable Plug Into Page*) virtually inserts a module image into the calculator, starting at the specified page. There are two options for specifying the module image and start page, and the function automatically distinguishes between the cases by examining the contents of the ALPHA register.

In the first case the ALPHA register holds both the four-character module identifier and

the page (in hexadecimal), separated by a space.

	ALPHA register					
	6	5	4	3	2	1
module identifier and page	***M4***	***M3***	***M2***	***M1***		***PG***

In the second case the ALPHA register holds the four-character module identifier and the X register holds the page (in decimal, 0-15).

	ALPHA register				X register
	4	3	2	1	
module identifier and page	***M4***	***M3***	***M2***	***M1***	***PG***

The **PPLUG** function distinguishes these two cases by examining characters 7-5 in the ALPHA register. If these three character locations are empty the page is sourced from the X register. Otherwise the page is sourced from character 0 in the ALPHA register.

In addition to the normal identifier/page combination, a number of other options are available, using the same formats. Only alphanumeric characters are allowed as the first or last character of a normal module identifier, and only a hexadecimal (or decimal, in the X register) number is allowed for the page character, so special characters or special identifiers are used to select the special cases for the **PPLUG** function.

An identifier of ***EMPT*** indicates that the image loaded in the specified page is to be unplugged. If the specified page is part of a multi-page image the entire image will be unplugged. Multi-page images are tagged as such in the MMU registers, which allows the entire image to be automatically removed. Using just the ALPHA register:

	ALPHA register					
	6	5	4	3	2	1
"EMPT" and page	***E***	***M***	***P***	***T***		***PG***

Or, using the X register to hold the page:

	ALPHA register				X register
	4	3	2	1	
"EMPT" and page	***E***	***M***	***P***	***T***	***PG***

The **PPLUG** function can also be used with a direct page address. If the first character of the identifier is a minus sign (**-**) a single-page non-banked image is plugged into the selected page. If the first character is the plus sign (**+**) a two-page non-banked image is plugged in starting at the selected page. If the first character is the dollar sign (**$**) a three-page non-banked image is plugged in starting at the selected page. If the first character is the percent sign (**%**) a four-page non-banked image is plugged in starting at the selected page. If the first character is the multiply sign (*****) a single-page banked image is plugged into the selected page. If the first character is the divide sign (**/**) a two-page banked image is plugged in starting at the selected page. The starting physical address of the image is specified directly in the other three characters for both of these options. Using just the ALPHA register:

	ALPHA register						
	6	5	4	3	2	1	
"-", physical address and page	**-**	**P5**	**P4**	**P3**		**PG**	one page
"+", physical address and page	**+**	**P5**	**P4**	**P3**		**PG**	two pages
"$", physical address and page	**$**	**P5**	**P4**	**P3**		**PG**	three pages
"%", physical address and page	**%**	**P5**	**P4**	**P3**		**PG**	four pages
"*", physical address and page	*****	**P5**	**P4**	**P3**		**PG**	one page, banked
"/", physical address and page	**/**	**P5**	**P4**	**P3**		**PG**	two pages, banked

Using the X register to hold the page:

	ALPHA register				X register	
	4	3	2	1		
"-", physical address and page	**-**	**P5**	**P4**	**P3**	**PG**	one page
"+", physical address and page	**+**	**P5**	**P4**	**P3**	**PG**	two pages
"$", physical address and page	**$**	**P5**	**P4**	**P3**	**PG**	three pages
"%", physical address and page	**%**	**P5**	**P4**	**P3**	**PG**	four pages
"*", physical address and page	*****	**P5**	**P4**	**P3**	**PG**	one page, banked
"/", physical address and page	**/**	**P5**	**P4**	**P3**	**PG**	two pages, banked

A question mark (**?**) in the first character of the module identifier field queries the MMU registers for the image loaded at the selected page. The remaining three characters of the module identifier are ignored in this case.

The image identifier (or the address, if no corresponding identifier can be found) and page are returned in the ALPHA register as well as the display. Using just the ALPHA register:

	ALPHA register					
	6	5	4	3	2	1
"?IMG" and page	**?**	**I**	**M**	**G**		**PG**

Or, using the X register to hold the page:

	ALPHA register				X register
	4	3	2	1	
"?IMG" and page	**?**	**I**	**M**	**G**	**PG**

Finally, a question mark (**?**) in place of the page character queries the MMU for the specified image or address. If the image or address is found in an MMU register marked as Valid, the identifier and page are returned in the ALPHA register and the display. This type of query can only be done using the ALPHA register:

	ALPHA register					
	6	5	4	3	2	1
module identifier and "?"	**M4**	**M3**	**M2**	**M1**		**?**
"-", physical address and "?"	**-**	**P5**	**P4**	**P3**		**?**

PLUG (prompts for image identifier and start page)

Executing **PLUG** (*Plug Into Page*) virtually inserts a module image into the calculator, starting at the specified page. This function is identical to the **PPLUG** function except that it prompts for both the image identifier and the page. This function is not programmable.

This function only accepts a minus sign (**-**) or a multiply sign (*****) along with a direct page address, so use the **PPLUG** function if you need to plug in something other than a single page along with a direct page address.

This function provides shortcuts for the unplug and query-by-page cases:

The sequence **ALPHA ALPHA** automatically enters ***EMPT*** for the module identifier, to streamline the unplug case.

The sequence **SHIFT ALPHA ALPHA** automatically enters ***?IMG*** for the module identifier, to streamline the query-by-page case.

The **PLUG** function allows one more optional feature that the **PPLUG** function does not support. A question mark (***?***) in both the first character of the identifier and the page character queries the MMU, displaying all of the images currently plugged in via the MMU. This query can be paused using the **R/S** key, and cancelled using the backspace key. The last result remains in the ALPHA register upon exit, as well as the display.

EXPG (prompts for source and destination page)

Executing **EXPG** (*Exchange Pages*) exchanges the MMU programming for two pages. This function should only be used to exchange single-page (banked or non-banked) images. Exchanging one page of a multi-page image will disrupt the MMU information that allows the automatic unplugging of multi-page images. Only pages 6 through F are valid for source or destination pages.

MVPG (prompts for source and destination page)

Executing **MVPG** (*Move Page*) replaces the MMU programming for the destination page with the MMU programming for the source page. The source page is then unplugged. This function should only be used to move a single-page (banked or non-banked) image. Moving one page of a multi-page image will disrupt the MMU information that allows the automatic unplugging of multi-page images. Only pages 6 through F are valid for source or destination pages.

Memory Block Functions

The memory block functions allow you to manipulate pages (4K blocks) of memory. In particular, a page can be initialized to a user-selected value or copied to another location in memory.

Because these functions are operating on 4096 memory locations, the 41CL is automatically switched to the 50x Turbo mode during the transfer, and the current Turbo mode is restored after the transfer or initialization is complete.

Neither of these functions check whether the blocks are in Flash memory or RAM, so if you attempt to write to Flash memory the operation will appear to proceed, but without any writes occurring.

YMCLR (address and data in ALPHA register)

Executing **YMCLR** (*Clear Memory Block*) writes the contents of the data field to an entire 4K block of RAM memory starting at the address specified in the address field.

The address field is truncated to create an address that is on a 4K boundary before the writes commence, and only memory addresses are valid for this function. The **YMCLR** function cannot be used to write to physical modules.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YMCLR** function.

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
physical address	P5	P4	P3	P2	P1	P0	-	D3	D2	D1	D0
logical address	L3	L2	L1	L0	-	B	-	D3	D2	D1	D0

YMCPY (starting address pair in ALPHA register)

Executing **YMCPY** (*Copy Memory Block*) copies the contents of one 4K block of memory (Flash or RAM) to another 4K block of memory (RAM only.) This function only

allows copying blocks of memory that start on 4K boundaries and are 4K in length and only memory addresses are valid for this function.

When executed from the keyboard a static ***COPYING*** message is written to the display during the actual transfers. These transfers are executed in 50x Turbo mode.

If the MMU is not enabled for a source logical address specified with the **YMCPY** function, the data is fetched from a physical module and copied to internal memory. In this case the bank identifier is ignored, because the physical module will be in control of the bank select. This means that only Bank 1 of a physical module can be copied to memory.

This function can be used to write to a physical Port. In this case the function uses the WROM instruction, which only writes 10 bits.

The **YMCPY** function is ideal for creating backups of system information such as the 41C register memory or the MMU contents. To backup the 41C register memory (including all user programs) simply copy the contents of memory starting at address 0x800000 to an available block of RAM. Use address 0x804000 to backup the MMU configuration.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YMCPY** function.

	ALPHA register						
	7	6	5	4	3	2	1
	source				destination		
physical address to physical address	P5	P4	P3	>	P5	P4	P3
physical address to logical address	P5	P4	P3	>	L3	-	B
logical address to physical address	L3	-	B	>	P5	P4	P3
logical address to logical address	L3	-	B	>	L3	-	B

Memory/IO Read and Write Functions

The entire memory space is accessible using these functions, which means that you can write directly to 41C register memory, program the MMU (which is not recommended), or modify (i.e. corrupt) Operating System variables.

YPOKE (address and data in ALPHA register)

Executing **YPOKE** (*Write Word to Memory or I/O*) writes directly to either RAM memory or an internal I/O port. This function does not check the address except for proper formatting, so attempting to write to Flash memory is allowed, although it will be ignored because the function does not properly format the write for Flash memory.

The peripheral version of this function only writes 12 bits to the peripheral port. This is okay because none of the peripheral write locations accept more than 12 bits.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YPOKE** function. A bank identifier of ***A*** in the logical address case allows writing the to all four banks simultaneously.

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
physical address	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***D3***	***D2***	***D1***	***D0***
logical address	***L3***	***L2***	***L1***	***L0***	***-***	***B***	***-***	***D3***	***D2***	***D1***	***D0***
port address				***R***	***-***	***-***	***-***	***D3***	***D2***	***D1***	***D0***

YPEEK (address and data in ALPHA register)

Executing **YPEEK** (*Read Word from Memory or I/O*) reads directly from either memory (Flash or RAM) or an internal I/O port. The data field in the ALPHA register when the function is called is ignored, but is replaced with the actual data read from either the memory or the internal I/O port.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YPEEK** function. The placeholder data characters must be present, and

will be replaced by the data read by the function.

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
physical address	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***D3***	***D2***	***D1***	***D0***
logical address	***L3***	***L2***	***L1***	***L0***	***-***	***B***	***-***	***D3***	***D2***	***D1***	***D0***
port address				***R***	***-***	***-***	***-***	***D3***	***D2***	***D1***	***D0***

Memory Buffer Functions

The 41CL reserves one page (one block of 4K words) of memory in RAM to be used for buffers. The Extra Functions Buffer Area is located at physical addresses 0x805000 - 0x805FFF, and there are three separate buffer pointers available to point to addresses in this area.

The three buffer pointers are stored at addresses 0x804010, 0x804012 and 0x804016. These buffer pointers provide a convenient way to move data to and from the buffer area without having to specify an address.

The *Memory Buffer Pointer*, stored at address 0x804010, is used for transfers from memory to the Buffer Area. The **YBUILD** function uses this buffer pointer to allow you to assemble a module image without having to continuously specify the destination address. Instead, the lower twelve bits of the destination address are held in the *Memory Buffer Pointer*.

To assemble blocks of code you merely initialize the *Memory Buffer Pointer* to the start of the block, with either 0x000 if assembling a FAT, or 0x084 if assembling functions, and then copy blocks of memory, one after the other, to the buffer. The buffer pointer is updated to point at the next buffer location after each copy. Once an image is assembled, the FAT can be built using the regular **YPOKE** function and the entire image moved to another location in memory using the **YMCPY** function.

The *Get Buffer Pointer*, stored at address 0x804016, is used for transfers from the serial port to the Buffer Area. The **YGETLB** and **YGETUB** functions use this buffer pointer to allow you to transfer received serial data to memory without having to continuously specify the destination address. Instead, the lower twelve bits of the destination address are held in the *Get Buffer Pointer*, which increments after each transfer.

The *Put Buffer Pointer*, stored at address 0x804012, is used for transfers from the Buffer Area to the serial port. The **YPUTLB** and **YPUTUB** functions use this buffer pointer to allow you to transfer memory data to the serial transmitter without having to continuously specify the source address. Instead, the lower twelve bits of the source address are held in the *Put Buffer Pointer*, which increments after each transfer.

Pointer Functions

YBPNT
YGPNT **(data in ALPHA register)**
YPPNT

Executing **YBPNT** (*Write Memory Buffer Pointer*) writes data directly to the *Memory Buffer Pointer* at address 0x804010.

Executing **YGPNT** (*Write Get Buffer Pointer*) writes data directly to the *Get Buffer Pointer* at address 0x804016.

Executing **YPPNT** (*Write Put Buffer Pointer*) writes data directly to the *Put Buffer Pointer* at address 0x804012.

The data for these commands must be a four-digit hex number, but only the lower three digits of this value are used as the buffer pointer. The most-significant digit is stored in memory, but is not changed by the buffer functions.

ALPHA register			
4	3	2	1
Buffer pointer value	***D3***	***D2***	***D1***	***D0***

These functions return with the normal **YPOKE** formatted physical address of the buffer pointer in the ALPHA register (but not the display):

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
Mem Buffer Pointer	***8***	***0***	***4***	***0***	***1***	***0***	***-***	***D3***	***D2***	***D1***	***D0***
Get Buffer Pointer	***8***	***0***	***4***	***0***	***1***	***6***	***-***	***D3***	***D2***	***D1***	***D0***
Put Buffer Pointer	***8***	***0***	***4***	***0***	***1***	***2***	***-***	***D3***	***D2***	***D1***	***D0***

YBPNT?
YGPNT?
YPPNT?

Executing **YBPNT?** (*Read Memory Buffer Pointer*) reads directly from the *Memory Buffer Pointer* at address 0x804010.

Executing **YGPNT?** (*Read Get Buffer Pointer*) reads directly from the *Get Buffer Pointer* at address 0x804016.

Executing **YPPNT?** (*Read Put Buffer Pointer*) reads directly from the *Put Buffer Pointer* at address 0x804012.

These functions all return with the normal **YPEEK** formatted physical address of the buffer pointer in the ALPHA register and the display.

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
Memory Buffer Pointer	***8***	***0***	***4***	***0***	***1***	***0***	***-***	***D3***	***D2***	***D1***	***D0***
Get Buffer Pointer	***8***	***0***	***4***	***0***	***1***	***6***	***-***	***D3***	***D2***	***D1***	***D0***
Put Buffer Pointer	***8***	***0***	***4***	***0***	***1***	***2***	***-***	***D3***	***D2***	***D1***	***D0***

YGPNT-
YPPNT-

Executing **YGPNT-** (*Decrement Get Buffer Pointer*) decrements the *Get Buffer Pointer* at address 0x804016.

Executing **YPPNT-** (*Decrement Put Buffer Pointer*) decrements the *Put Buffer Pointer* at address 0x804012.

The serial transfer functions that use buffer pointers all increment the buffer pointer after each byte transfer. This is fine if you want to transfer just one byte per memory location, but if packing two bytes per memory word is required (as will normally be the case) these decrement functions can be used after alternate byte transfers to pack two bytes into each memory word.

Data Transfer Functions

YBUILD (starting address and transfer length in ALPHA register)

Executing **YBUILD** (*Write to Extra Functions Buffer*) copies a block of data (up to 4096 words) from memory to the Extra Functions Buffer Area, starting at the location addressed by the *Memory Buffer Pointer*. The buffer pointer is updated to point at the next Buffer Area location at the end of the transfer.

When executed from the keyboard a static ***COPYING*** message is written to the display during the actual transfers. The transfers are executed in 50x Turbo mode, and then the current Turbo mode is restored.

The **YBUILD** function only supports physical addresses. This means that if you want to transfer data from a physical module to the Buffer Area the data must first be transferred to RAM memory so that a physical address can be specified.

The figure below shows the formatting required for the address and data for the **YBUILD** function. The transfer length ***D2*** - ***D0*** is limited to 4096 words or less, and the number of words transferred is the transfer length. ***000*** indicates a transfer length of 4096 words. Be careful, because this function will wrap around the end of the Buffer Area, back to the beginning of the Buffer Area, if the transfer length specified so indicates.

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
physical address	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***0***	***D2***	***D1***	***D0***

YGETLB
YGETUB

Executing **YGETLB** (*Write Serial Byte to Lower Memory Byte*) reads one byte from the serial port and writes this byte to the lower byte of the memory location specified by the *Get Buffer Pointer* and then increments this pointer.

Executing **YGETUB** (*Write Serial Byte to Upper Memory byte*) reads one byte from the serial port and writes this byte to the upper byte of the memory location specified by the *Get Buffer Pointer* and then increments this pointer.

These functions put a ***RECEIVING*** message in the display while waiting for a character. If the serial port encounters an overrun condition the data in the receive buffer

is discarded (since it is in error anyway) and is not written to memory.

YPUTLB
YPUTUB

Executing **YPUTLB** (*Write Lower Memory Byte to Serial Por*t) reads the lower byte from the address specified by the *Put Buffer Pointer* and attempts to write it to the serial port. If this transfer is successful the pointer then increments.

Executing **YPUTUB** (*Write Upper Memory Byte to Serial Por*t) reads the upper byte from the address specified by the *Put Buffer Pointer* and attempts to write it to the serial port. If this transfer is successful the pointer then increments.

These functions put a ***SENDING*** message in the display while waiting to send a character.

All of the serial data transfer functions work independent of the mode (synchronous or asynchronous) of the serial port, and all of the functions contain a time-out feature to prevent locking up the machine in the case of an unavailable serial port or unresponsive device on the other end of the serial interface. This time-out period is dependent on the Turbo mode, as shown in the table below:

Speed	Serial time-out period
1x	~750mS
2x	~600mS
all others	~350mS

Flash Memory Functions

If you are not absolutely sure of what you are doing, do not attempt to use these functions! While these functions do prevent you from corrupting the Operating System of the calculator, they still allow you to erase or modify the rest of the Flash memory. You must be familiar with how Flash memory operates before attempting to use these functions.

Flash memory has limited endurance, typically 100,000 write cycles, and is erased by sectors, which are 64K bytes (32K words, or eight pages) in the case of the 41CL. An erased Flash sector returns 0xFFFF in every location. Only 0's can be written to any given location in Flash, which means that writes to Flash can only change a "1" to a "0" and never vice-versa.

During a Flash erase or write, no other Flash memory accesses are allowed. This means that these functions must be resident in RAM to work. So, if you really want to use either of the Flash Memory functions you must copy the entire *41CL Extreme Functions* image to RAM and then program the MMU to use this RAM copy of these functions. Starting with revision -2A of the *41CL Extreme Functions*, this copy to RAM is not necessary, because each function automatically copies a small section of code to RAM and then executes the code in RAM. However, if you are erasing or writing to the sector that contains the *41CL Extreme Functions*, you must still use a RAM copy of the code.

YFERASE (address in ALPHA register)

Executing **YFERASE** (*Erase FlashMemory Sector*) erases an entire sector (usually 32K words, or eight pages) of Flash memory. The address specified can lie anywhere within the sector.

The **YFERASE** function automatically includes a 6 second delay, because the Flash erase operation may require this much time to complete. The function will either return immediately with an error message, without executing, or send the ***ERASING*** message to the display for the entire 6 seconds before returning.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YFERASE** function.

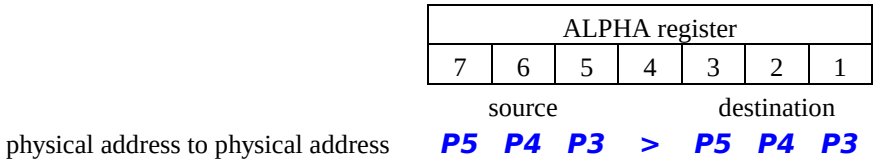
ALPHA register					
6	5	4	3	2	1
physical address	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***

YFWR (starting address pair in ALPHA register)

Executing **YFWR** (*Write Flash Memory Page*) copies the contents of one 4K word block (one page) of RAM memory to a 4K word block of Flash memory. This function only allows copying blocks of memory that start on 4K boundaries and are 4K in length. Only physical memory addresses are valid for this function.

The **YFWR** function automatically executes at 50x Turbo speed, but still requires approximately 4 seconds to complete. The current Turbo mode is restored when the function completes. The function will either return immediately with an error message, without executing, or send the ***WRITING*** message to the display for the entire 4 seconds before returning.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YFWR** function.



Serial Port Functions

The 41CL contains a dual-mode serial port. In asynchronous mode RS-232 is supported, but the special PCB connector is required to bring the signals out of the calculator. In synchronous mode the serial port can be used to communicate with an SD card, but this requires soldering several wires directly to the 41CL board, because the signals are not brought to a connector. At this time only RS-232 communication is supported, so only that operation will be described here.

The hardware is initialized in asynchronous mode whenever the calculator is turned on. The serial port uses 8N1 format (eight bits of data, no parity, and one stop bit).

Depending on the baud rate, it is usually advisable to run the 41CL in 50x Turbo mode when performing serial operations to make sure that the CPU has sufficient speed to keep up with the serial port. The serial port functions do not automatically increase the processor speed to 50x.

Even using the 50x Turbo mode, at higher baud rates there will be gaps between transmit characters and the receiver will need gaps between receive characters. This is because some instructions still run at 1x speed independent of the Turbo mode. Keep this restriction in mind when using the serial block transfer functions. If the source of serial data generates serial characters without any intervening idle time it will probably be necessary to use 1200 baud to prevent receive overruns.

The serial functions only support physical addresses. This means that if you want to transfer data between a physical module and the serial port the data must be buffered in RAM memory before the final transfer to or from the physical module.

All of the serial data transfer functions contain a time-out feature to prevent locking up the machine in the case of an unavailable serial port. This time-out period is dependent on the Turbo mode, as shown in the table below:

Speed	Serial time-out period
1x	~15 seconds
2x	~12 seconds
all others	~7 seconds

In the absence of a valid RS-232 level on the receive input the RS-232 transceiver automatically powers down. But whenever there is a valid RS-232 level on the receive input the transceiver will be powered. This is a significant addition to the current drain on the batteries, so the serial port should only be connected to a PC or other RS-232 equipment when actually using the serial port.

The calculator should always be turned off while connecting or disconnecting the

serial port. The recommended way to connect the serial port is to first insert the 2.5mm plug into the calculator and then connect the other end to an active serial connection. While the serial driver in the calculator is powering up the internal power supply may droop low enough to trigger the power-on-reset, which automatically disables the MMU. This droop is not sufficient to corrupt RAM contents, so the MMU programming will still be valid. So, after turning on the calculator with the serial port connected for the first time, it is advisable to make sure that the MMU is enabled before attempting to use the serial functions.

The tension holding a 2.5mm plug in the serial connector jack is higher than the tension holding the blank port cover in the calculator body. This means that trying to pull out the plug will tend to pull the blank port cover out of the calculator, potentially damaging the internal connections to the serial connector jack. **Always remember to hold the blank port cover in place when attempting to remove the serial port plug from the calculator.**

RS-232 Control Functions

SERINI

Executing **SERINI** (*Initialize Serial Port*) initializes the serial port in asynchronous mode and sets the baud rate to 1200. Both the transmit and receive buffers are emptied and the receiver and transmitter are both set to the idle state. This command disables the RS-232 driver, unless a valid RS-232 level is present in the receive input.

SERON

Executing **SERON** (*Enable Serial Port Driver*) forces the RS-232 driver on, independent of the state of the RS-232 receive input. This command will be necessary when connecting to another RS-232 port that only powers up with a valid RS-232 level on its receive input (as will be the case when connecting to another 41CL.) Note that this command will not work on V2 or V3 boards, as they lack the requisite connection between the NEWT processor and the RS-232 driver on the board.

PBAUD (baud rate in X register)

Executing **PBAUD** (*Programmable Baud Rate Select*) sets the Baud Rate for the serial port to the value in the X register. Only the values 12 (1200 baud), 24 (2400 baud), 48 (4800 baud), and 96 (9600 baud) are allowed.

BAUD prompts for two-digit value

Executing **BAUD** (*Baud Rate Select*) prompts the user for a two-digit Baud Rate value. Only the values 12 (1200 baud), 24 (2400 baud), 48 (4800 baud), and 96 (9600 baud) are allowed. This function is not programmable.

Block Transfer Functions

YEXP (address and transfer length in the ALPHA register)

Executing **YEXP** (*Export Memory Block*) transfers an entire block of data (up to 4096 words) from internal memory to the serial port. Both Flash and RAM addresses are valid for this function.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YEXP** function. The transfer length is limited to 4096 (words) or less. The number of words transferred is the transfer length plus one, allowing block transfers of from 1 to 4096 words (2 to 8192 bytes).

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
physical address	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***0***	***D2***	***D1***	***D0***

This function puts a static ***SENDING*** message in the display while waiting to send characters.

The **YEXP** function transfers words one byte at a time, in little-endian order (least significant byte first), from the lowest memory address (the one specified in the ALPHA register) to the highest memory address.

In case of an error the transfer length in the ALPHA register will be updated to show the

number of words remaining to be transferred. This allows the function to be started again without modifying the contents of the ALPHA register. Only the transfer of both bytes of a word counts as a successful transfer. If the transfer times out between the first and second byte of a word transfer, the transfer is deemed unsuccessful.

YIMP (address and transfer length in the ALPHA register)

Executing **YIMP** (*Import Memory Block*) transfers an entire block of data from the serial port into internal memory. The address must be in RAM, because this function does not properly format writes for the Flash memory. Transferring data to a physical Port is not supported either.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YIMP** function. The transfer length is limited to 4096 (words) or less. The number of words transferred is the transfer length plus one, allowing block transfers of from 1 to 4096 words (2 to 8192 bytes).

	ALPHA register										
	11	10	9	8	7	6	5	4	3	2	1
physical address	***P5***	***P4***	***P3***	***P2***	***P1***	***P0***	***-***	***0***	***D2***	***D1***	***D0***

This function puts a static ***RECEIVING*** message in the display while waiting for a character. If the serial port encounters an overrun condition the data in the receive buffer is discarded (since it is error anyway) and is not written to memory.

The **YIMP** function transfers words one byte at a time, in little-endian order (least significant byte first), from the lowest memory address (the one specified in the ALPHA register) to the highest memory address.

In case of an error the transfer length in the ALPHA register will be updated to show the number of words remaining to be transferred. This allows the function to be started again without modifying the contents of the ALPHA register. Only the transfer of both bytes of a word counts as a successful transfer. If the transfer times out between the first and second byte of a word transfer, or the receiver overflows on either byte of a word transfer the transfer is deemed unsuccessful.

Synchronous Mode Control Functions

SYNINI

Executing **SYNINI** (*Initialize Synchronous Port*) enables the serial port in synchronous mode, but does not activate the CE# (chip enable) signal. This function does not reset the serial port, so **SERINI** should be executed first, to completely reset the serial port hardware.

SYNON

Executing **SYNON** (*Enable Synchronous Port*) enables the serial port in synchronous mode, and activates the CE# (chip enable) signal. An active CE# signal is required to read or write an attached clocked serial device, such as an SD card.

SYNDIV (divider in X register)

Executing **SYNDIV** (*Synchronous Port Divisor Select*) programs the clock rate divider for the serial port directly, with the divisor value in the X register. This function is primarily for use with synchronous mode, where the serial bit rate will be 9MHz divided by “n+1” (the divisor value, plus one.)

This function can also be used to create non-standard baud rates for RS-232 communication. In this case the clock rate divider output frequency is 18MHz divided by “n+1”. This output frequency is further divided by sixteen to create the baud rate. For example, 1200 baud requires a divisor of 936, 2400 baud requires a divisor of 467, 4800 baud requires a divisor of 233, and 9600 baud requires a divisor of 116.

Miscellaneous Functions

YFNS?

Executing **YFNS?** (*Read 41CL Extreme Functions Location*) polls for the current location of the 41CL Extreme Functions. The page where the 41CL Extreme Functions reside is returned in the X register as a decimal number in the range ***6*** through ***15***, corresponding to Pages 6 through F.

CHKXROM

Executing **CHKXROM** (*Check for XROM Conflicts*) polls pages 6 through F, searching for duplicate XROM numbers. If no duplicate is found the function returns with an ***XROM OK*** message in the display. If a duplicate XROM number is found the function returns with an ***XROM ERROR*** message in the display.

Note that some multi-page module images loaded in the 41CL use the same XROM number for more than one page, and will be flagged as a result. XROM conflicts are only a problem when using image functions in a program.

YCRC (3-digit Page address in ALPHA register)

Executing **YCRC** (*Calculate Page CRC*) calculates the 32-bit Cyclic Redundancy Check (CRC) used for Ethernet over the entire 4K words of a memory page. The calculated CRC will be unique for each page, and is capable of detecting all burst errors up to 32 bits in length.

The **YCRC** function uses a three-digit address of a page in physical memory to select which page to operate on.

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

physical page address

The CRC result is returned in the ALPHA register (overwriting the page address) as well

as the display:

ALPHA register							
8	7	6	5	4	3	2	1
calculated CRC	***D7***	***D6***	***D5***	***D4***	***D3***	***D2***	***D1***	***D0***

The **YCRC** function automatically executes in 50X Turbo mode, but still requires several seconds to complete. The execution time is data-dependent, with a minimum execution time of about 4.5 seconds and a maximum execution time of about 12 seconds. This function puts a static ***WORKING*** message in the display while running.

Image Database Functions

The Image Database Functions allow you to search, modify, add to, and store the Image Database. Although the Image Database normally resides in Flash memory, it is also possible to operate on a copy of the Image Database that has been stored in the Extra Functions Buffer Area of RAM. There is no default selection of Flash or RAM, so either the **IMDBF** command or the **IMDBR** command must be issued before using the Image Database.

PIMDB? (module identifier or Page address in ALPHA register)

Executing **PIMDB?** (*Programmable Image Database Search*) searches the selected Image Database for a match, using the either a module identifier or a page address, and returns the corresponding database information.

The figure below shows the formatting for a module identifier or page address:

	ALPHA register			
	4	3	2	1
module identifier	***M4***	***M3***	***M2***	***M1***
page address	***-***	***P5***	***P4***	***P3***

The Image Database information is returned in both the ALPHA register and the display, in the format shown in the figure below. Like the **PLUG** and **PPLUG** functions, only the first and last characters of the module identifier are used to search the Image Database, but the data returned by the **PIMDB?** function contains all four characters of the module identifier found in the Image Database.

	ALPHA register									
	10	9	8	7	6	5	4	3	2	1
physical address	***M4***	***M3***	***M2***	***M1***	***-***	***T***	***-***	***P5***	***P4***	***P3***

The type digit ***T*** specifies the type of image, according to the table below.

T digit	image type for this module identifier
0	4K image (one page)
1	8K image (two pages)
2	16K image (all four banks in one page)
3	16K (four pages)
4	32K (all four banks in two pages)

Since a module image may be up to 32K in length (8 pages), more than one physical address can return with a match, but only the information in the actual database entry is returned. Only the first match (the search proceeds from lowest database address upwards) will ever be returned. Searches with a valid module identifier always return the corresponding contents of the Image Database, even if the entry is unprogrammed.

This function automatically executes the search in the 50X Turbo mode, but even so the search may take several seconds when searching for an address match. A static ***SEARCHING*** message is written to the display while a search is in progress.

The Image Database loaded into the 41CL during manufacturing contains more information than is returned by the **IMDB?** function. This extra information includes the module group, any page restrictions, and special modifiers for the type digit.

IMDB? (prompts for module identifier or page address)

Executing **IMDB?** (*Image Database Search*) searches the selected Image Database for a match, using the either a module identifier or a page address, and returns the corresponding database information. This function is identical to the **PIMDB?** function except that it prompts for the image identifier or page address. This function is not programmable.

This function also allows two optional features that the **PIMDB?** function does not support, as shown below:

	ALPHA register			
	4	3	2	1
module identifier with last character of "?"	***M4***	***M3***	***M2***	***?***
"?" and Image Group	***?***	***G3***	***G2***	***G1***

A question mark in place of the last character of a module identifier queries the Image Database for valid entries starting with the ***M4*** character. Only the module identifier (without the type and physical address) is returned by this query, so that the identifier can be used directly by the **PPLUG** function:

ALPHA register			
4	3	2	1

module identifier ***M4 M3 M2 M1***

A question mark in the first character of the identifier, along with a three-character module group, lists all of the module identifiers in the Image Database tagged as the selected group. Supported groups are shown in the table below.

AST	Astronomy
AVI	Aviation
CHM	Chemistry
ENG	Engineering
FIN	Financial
GAM	Games
GEN	General-purpose
GOV	Government/Military
HIL	HP-IL
HWS	Hardware-specific
MAT	Mathematics
MED	Medicine
NAV	Navigation
NUL	Nulled Entry
OSL	OS/CL
PHY	Physics
PRG	Programming
SVY	Surveying
SYS	System extensions
UTL	Utilities
UNP	Unprogrammed

Only the module identifier (without the type and physical address) is returned by this query, so that the identifier can be used directly by the **PPLUG** function:

ALPHA register			
4	3	2	1

module identifier ***M4 M3 M2 M1***

Both of these Image Database queries normally continue returning results until halted with the **R/S** key. This listing can be stopped and started using the **R/S** key. The function is cancelled using the backspace key (only while the listing is halted.) **SST** can be used to step through entry-by-entry, and **BST** steps through entry-by-entry in the reverse direction. The last result remains in the ALPHA register upon exit, as well as the display.

IMDBF

Executing **IMDBF** (*Image Database in Flash Memory*) sets an internal flag so that the **IMDB?** and **IMDBINS** functions will use address 0x0DF000 in Flash memory as the location of the Image Database.

There is no default selection of Flash or RAM, so this command (or the **IMDBR** command) must be issued before using either the **IMDB?, PIMDB?** or **IMDBINS** functions.

IMDBR

Executing **IMDBR** (*Image Database in RAM*) sets the internal flag so that the **IMDB?** and **IMDBINS** functions will use the Extra Functions Buffer area as the location of the Image Database. All other 41CL Extra Functions, including the **PLUG** functions, always use the Image Database residing in the Flash memory.

IMDBF?

Executing **IMDBF?** (*Test Image Database Location*) tests the Image Database location flag, returning with ***NO*** in the display if the RAM version of the Image Database is selected and ***YES*** in the display if the Flash version of the Image Database is selected.

When **IMDBF?** is used in a program, if the Flash version of the Image Database is selected the next program line will be executed; and if the RAM version of the Image Database is selected the next line in the program is skipped.

IMDBCPY

Executing **IMDBCPY** (*Copy Image Database to RAM*) copies the Image Database at address 0x0DF000 in Flash memory to the Extra Functions Buffer area.

This function automatically executes in 50x Turbo mode and is equivalent to **YMCPY** with ***0DF>805*** in the ALPHA register.

IMDBUPD

Executing **IMDBUPD** (*Update Image Database in Flash*) writes the contents of the Extra Functions Buffer area to Flash memory starting at address 0x0DF000.

This function is equivalent to **YFWR** with ***805>0DF*** in the ALPHA register.

IMDBINS (module identifier, type, and starting address in ALPHA register)

Executing **IMDBINS** (*Insert Image Database Entry*) inserts a database entry at the appropriate location in the Image Database.

The table below shows the formatting required for the module identifier, type, and address for the **IMDBINS** function.

	ALPHA register									
	10	9	8	7	6	5	4	3	2	1
physical address	***M4***	***M3***	***M2***	***M1***	***-***	***T***	***-***	***P5***	***P4***	***P3***

Image Database entries written using the **IMDBINS** function include only the information shown here. All of the special information about image group, page restrictions and type modifiers is written with the appropriate default values.

When using the copy of the Image Database in RAM the new entry is unconditionally written to the appropriate location. When writing to the Image Database in Flash memory the existing database entry should be unprogrammed, or the entry will likely be corrupted by the write. Writing 0 to the type field, along with an address of 000 will always create a null entry.

Error Conditions

All functions in the *41CL Update Functions* that require an argument are capable of generating multiple errors, but only the first error found will be reported. Functions that need one or two page addresses check for errors in the following order: format (two-address functions only), valid hexadecimal, address(es) within range, ending address greater than starting address, valid sector (sector functions only), destination is in RAM (if appropriate) or Flash (if appropriate), and source is in RAM (if appropriate) or Flash (if appropriate).

Functions requiring only one page address treat that address as a destination as far as error messages are concerned.

Arguments are always checked before any communication takes place, so the communications-related errors can only be generated if the arguments are deemed correct. The timeout limit of 30 seconds does not roll over from one byte to the next, but is restarted for each byte.

All error conditions cause a function to abort. Those functions that operate repetitively (**FDBCHK?**, **FLCHK?**, and **FLUPD**) will update the beginning address in the ALPHA register in the case of an error, to indicate how much progress was made towards completion.

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

Error Message	Function	Meaning
ADDR ERROR	**YFERASE** **YFWR**	Address not in Flash
BAD ID	**IMDB?** **IMDBINS** **PLUG** **PPLUG**	Invalid character in mnemonic
DATA ERROR	All communication functions using hex	Invalid hex digit
	BAUD **PBAUD**	Invalid baud rate
	PTURBO **TURBO**	Invalid turbo rate
	All functions requiring an address	Invalid address or format

LOCK ERROR	**EXPG** **MVPG** **PLUG** **PPLUG**	Page is locked
NO ENTRY	**PLUG** **PPLUG**	Unprogrammed IMDB entry
NO IMDB	**IMDB?** **PIMDB?**	Missing IMDB
NO MATCH	**PLUG** **PPLUG**	Could not find matching IMDB entry
NULL ENTRY	**PLUG** **PPLUG**	Null (erased) IMDB entry
OS AREA	**YFERASE** **YFWR**	Attempted operation on Operating System area of Flash
OVERRUN	Serial receive functions	Receiver overrun
PAGE ERROR	**EXPG** **MVPG** **PLUG** **PPLUG**	Image cannot be legally plugged into selected page
SIZE ERROR	**PLUG** **PPLUG**	Image will not fit in remaining pages
SRC=ROM	**YFWR**	Source address in Flash
TIMEOUT	Serial receive functions	No receive data in timeout period
TYPE ERROR	**PLUG** **PPLUG**	Image is not pluggable

Internal Details

The table below shows the XROM numbers for the programmable functions in the *41CL Extreme Functions*.

Function	XROM Number	Function	XROM Number
MMUDIS	XROM 15,01	SERON	XROM 15,38
MMUEN	XROM 15,02	YIMP	XROM 15,39
MMU?	XROM 15,03	YEXP	XROM 15,40
MMUCLP	XROM 15,04	SYNDIV	XROM 15,41
MMUCLS	XROM 15,05	SYNINI	XROM 15,42
PPLUG	XROM 15,06	SYNON	XROM 15,43
LOCK	XROM 15,08	YGPNT	XROM 15,44
UNLOCK	XROM 15,09	YGPNT?	XROM 15,45
LOCK?	XROM 15,10	YGPNT-	XROM 15,46
MAPDIS	XROM 15,16	YGETLB	XROM 15,47
MAPEN	XROM 15,17	YGETUB	XROM 15,48
PTURBO	XROM 15,19	YPPNT	XROM 15,49
TURBO?	XROM 15,21	YPPNT?	XROM 15,50
YPOKE	XROM 15,22	YPPNT-	XROM 15,51
YPEEK	XROM 15,23	YPUTLB	XROM 15,52
YMCLR	XROM 15,24	YPUTUB	XROM 15,53
YMCPY	XROM 15,25	YCRC	XROM 15,55
IMDBF	XROM 15,26	YFERASE	XROM 15,56
IMDBR	XROM 15,27	YFWR	XROM 15,57
IMDBF?	XROM 15,28	CHKXROM	XROM 15,58
PIMDB?	XROM 15,29	YFNS?	XROM 15,59
IMDBCPY	XROM 15,31	YBPNT	XROM 15,60
IMDBINS	XROM 15,32	YBPNT?	XROM 15,61
IMDBUPD	XROM 15,33	YBUILD	XROM 15,62
PBAUD	XROM 15,35	41CL	XROM 15,63
SERINI	XROM 15,37		

The *41CL Extreme Functions* use dedicated locations in RAM to store information required for proper operation. The table below lists these locations and their use.

address	Usage
0x804010 - 0x804011	**Memory Buffer Pointer**
0x804012 - 0x804013	**PUT Buffer Pointer**
0x804014	**IMDB location, IMDB? search type**
0x804015	**IMDB? search pointer**
0x804016 - 0x804017	**GET Buffer Pointer**
0x804018	**Turbo mode storage location**
0x804019	**IMDB? search group**
0x804030	**Dynamic page staorage**
0x804800 - 0x8048FF	**code during dynamic paging**
0x804F00 - 0x804F1F	**Temporary storage for XROM status**
0x804FFC - 0x804FFF	**IMDB micro-buffer**

Revision History

04/19/2017 Cleaned up and reorganized

