41CL Calculator

Acknowledgements:

This project never could have succeeded without **Warren Furlow**'s excellent Web site hp41.org. And even more important was his SDK41R6 software suite, for code development, and his V41 program for code debugging. Numerous people have answered my dumb questions on the Web site hpmuseum.org, and the book "Inside the HP-41" by **Jean-Daniel Dodin** was invaluable for getting a foothold on understanding the HP-41 operating system and register usage. **Gene Wright** was kind enough to be my voice at the HHC 2010 conference. **'Angel Martin** contributed valuable 41CL-specific software to the project. **Eric Rechlin** made the labels for the front of the calculator. **Nate Martin** modified his Port cover 3-D model specially for use with the serial port jack.

Table of Contents

Introduction

The 41CL takes advantage of modern technology to significantly add to the capabilities of the 41C system. In particular, the 41CL provides the following features:

- All features of an HP-41CX except for the Time Module. CX Time functions (the software) are included, but a Time module plugged into a Port is required for full timer functionality.
- Full 600-register Extended Memory is built in.
- Over 180 plug-in module images are built in. Functions are included to allow these images to be virtually plugged into a calculator Port and unplugged from a calculator Port.
- Turbo mode, which allows the calculator to run at up to 50X normal speed. Actual values available are 2X, 5X, 10X, 20X and 50X.
- Over 190 empty pages (4K in length) of Flash memory are available for non-volatile storage.
- 122 pages (4K in length) of RAM are available. All RAM is continuously powered.
- A sophisticated Memory Management Unit (MMU) allows full access to the large physical memory.
- Full bus compatibility for the Ports, allowing the use of any peripheral designed for the HP-41 system.
- A full-duplex serial port is available when the optional serial connector is used. This optional connector uses a 2.5mm stereo jack mounted in a blank port cover.

With these features, however, come some drawbacks:

- Power consumption is higher, at least while the calculator is off or in light sleep (between keystrokes). Where the original HP-41 required about 10uA while off, the 41CL requires about 110uA. This will lead to reduced battery life.

- The original HP-41 could retain memory contents for several minutes while the batteries were changed. Because of the higher current consumption, the 41CL only retains the memory contents for a few seconds while the batteries are out. For this reason, you should probably have an extra battery holder ready to go when changing batteries.

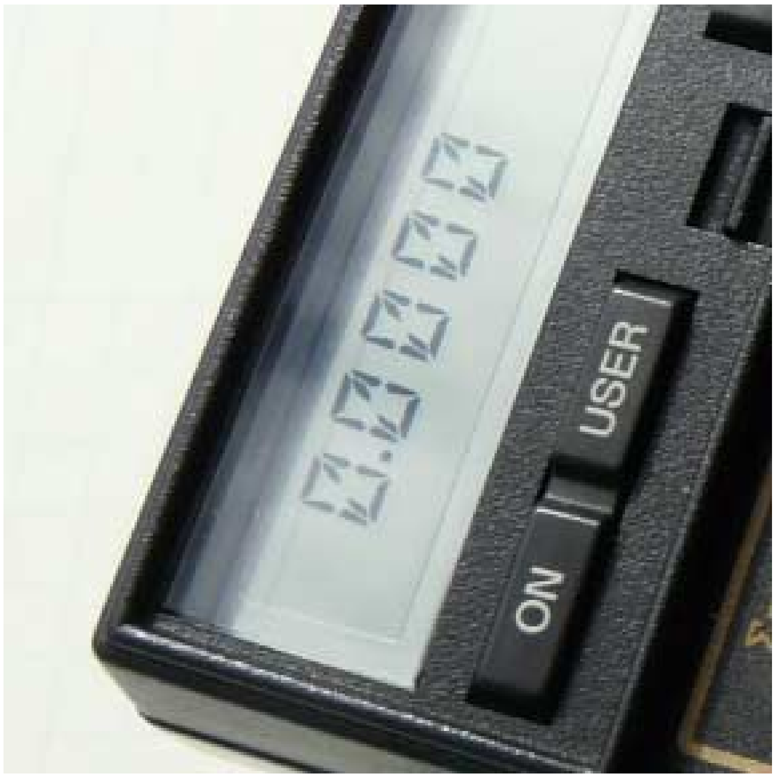
- The advanced technology used in the 41CL is a double-edged sword. The Flash memory, as well as the programmable logic devices used to implement the NEWT microprocessor, only guarantee data retention for 20 years.

The table below shows the typical current drain for the 41CL with and without the serial port powered up.

State	Serial port powered down	Serial port powered up
Off	110uA	2.8mA
Light Sleep (between key presses)	4.2mA	6.9mA
Running (1x)	7.9mA	10.6mA
Running (50x)	11.3mA	14.0mA

Can my calculator be upgraded?

The 41CL is an upgrade created by replacing the CPU circuit board in a 41C/CV/CX with the 41CL circuit board. This replacement is only possible for calculators that actually have a CPU circuit board. The easiest way to tell if this is the case is to look at the HP-41 display. If the light part of the display has square corners, like those shown below, the calculator is a candidate for replacing the CPU circuit board.



Hewlett-Packard changed the display driver circuitry in the 41 series during production, and this change affected one component value on the CPU circuit board. The 41CL circuit board implements the component value used in later production units. Units with the correct display driver can be identified as follows:

- If your 41C (not CV or CX) has a serial number starting with “1954” or larger it uses the correct display driver.

- If your 41CV/CX has a serial number starting with “2003” or larger it uses the correct display driver.

It is theoretically possible to use the 41CL circuit board with the older display driver, but this requires soldering an extra capacitor to either the 41CL circuit board or to the calculator main board. If you really want to upgrade a calculator that uses the older display driver, please contact us directly for details.

In addition to the display driver change, Hewlett-Packard also experimented with a different method of connecting the CPU board to the main board. Unfortunately it is not possible to identify units that used this different connection method via the serial number. Identifying such a calculator is a step in the installation process covered in the next section.

Getting Started

The 41CL circuit board is designed to be a drop-in replacement for the original CPU circuit board, and the installation is not difficult. However, certain precautions must be taken to prevent damage to both old and new circuitry.

All integrated circuits are susceptible to damage from electrostatic discharge (ESD). If you have access to an electrostatically protected work area by all means use it. If not, make sure that you ground yourself immediately before starting the installation process. The best way to keep from generating a static charge is to not move around while working, so make sure you have everything required before starting the process. **Do not touch exposed conductors, and handle both the original CPU circuit board and the 41CL circuit board by the edges.** The 41CL circuit board has a 2mm space around the edges devoid of circuitry or components to facilitate handling in this manner.

Tools required for the installation are a small Phillips-head screwdriver, an Xacto knife or similar, a pair of tweezers, and a small flashlight.

Installing the 41CL circuit board

Follow the steps below to perform the installation:

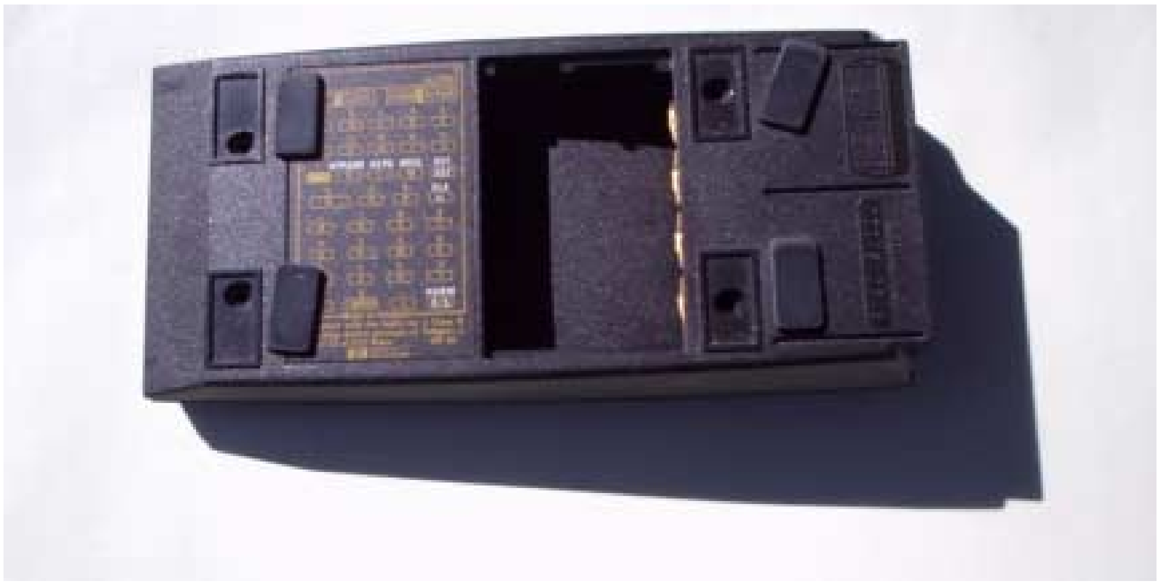
1. Read through all of these instructions before starting the installation process, to make sure that you understand each step. **We are not responsible if you damage or ruin your calculator or the 41CL circuit board while attempting this installation.**

2. Verify that your calculator is one that has a CPU circuit board. Only calculators with “square corners” on the LCD display panel (refer to the Introduction section for a picture) have CPU circuit boards.

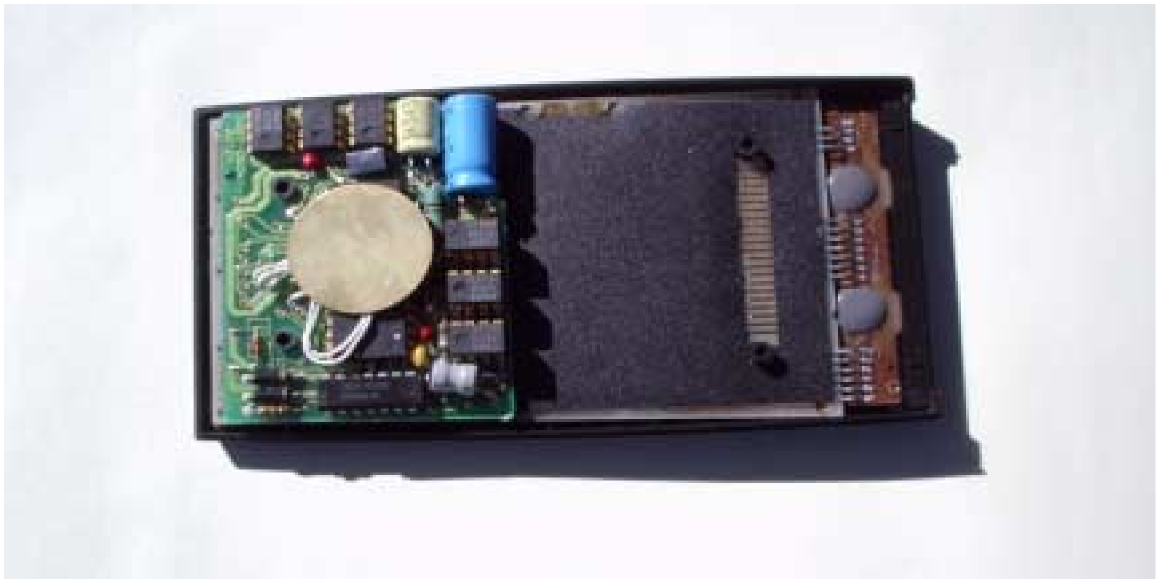
3. Remove the battery case, by first sliding the case towards the top of the calculator until the bottom end of the battery case pops free. Install fresh batteries.

4. Carefully remove the four rubber feet, using a pointed knife to pry up one corner of a foot and a pair of tweezers to lift the foot from the case. Be careful not to damage the feet, as replacements are difficult to find. They are attached to the calculator body using double-sided tape which can usually be reused.

5. Remove the four screws located in the recesses under the rubber feet. Be very careful not to lose them, as they are essentially impossible to find unless you want to buy 10,000 of them. The screws are all #2-28 trilobular thread-forming types. Those at the bottom of the case are 1/4” (they may be 3/8” if the calculator has been serviced), while those at the top of the case are 3/4”.



6. Lift off the bottom case and the U-shaped center case section. Note the orientation (front-to-back) of the center case section, because it not symmetric.

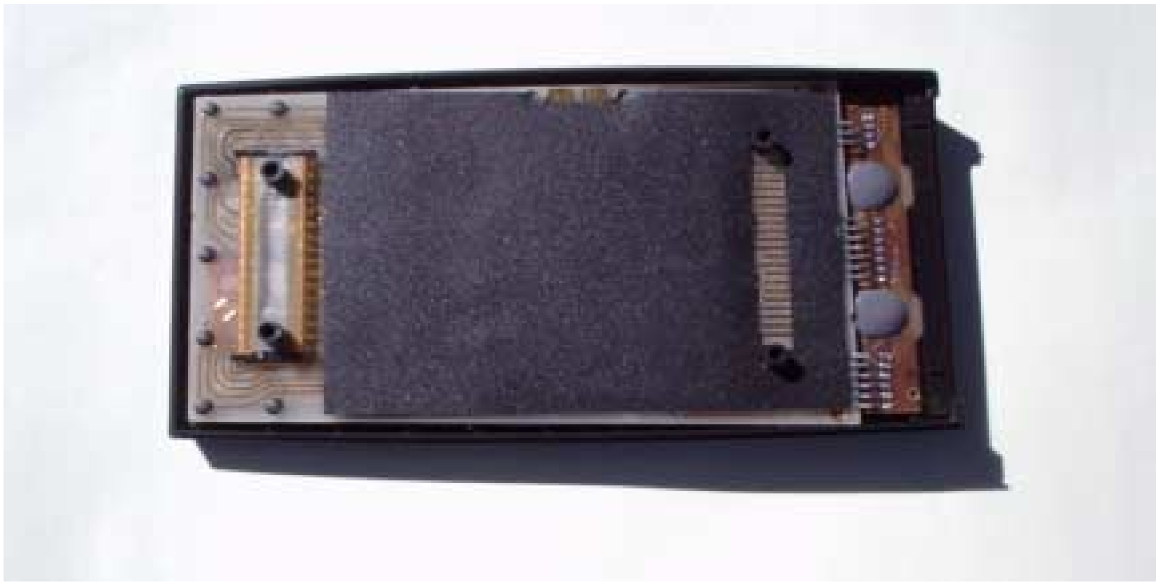


Don’t worry if your old CPU looks slightly different from that shown in the picture. These boards went through several revisions during the life of the 41 series.

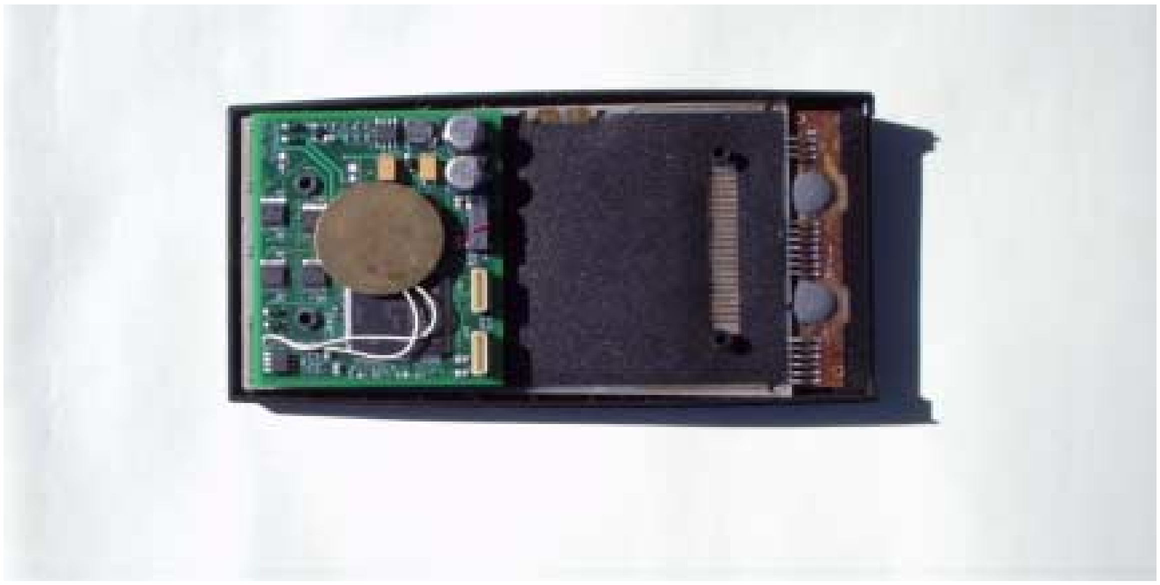
7. **STOP!** Before lifting the old CPU circuit board use a flashlight to look into the space between the CPU circuit board and the base circuit board. You will be able to see which type of connector is being used. The connector style in early production units will look like styrofoam. If this is the case, attempting to replace the CPU circuit board is not advised, as this type of connector material is not really reusable. Later production units use conductors rolled around a flexible plastic tubular form, which will appear shiny in the light from the flashlight. Being reusable, this type of conductor is suitable for use with the 41CL circuit board.

8. Before removing the old CPU circuit board, carefully inspect all four screw posts on the front case. If they are cracked or broken (a common problem) they will need to be repaired before re-assembling the calculator. Instructions for this repair can be found on the Museum of HP Calculators web site. Repair the screw posts before proceeding any further.

9. Using only the edges of the board, carefully lift the old CPU circuit board off of the base circuit board. The connectors will usually remain in place on the main circuit board. If by chance they do not, restore them to their proper location. The two halves of the connector are usually held together by a flexible piece of plastic that fits over the two screw posts just like the CPU circuit board. Some connectors were held in place by the black spacer hooking down around the ends of the connectors by about 1cm. This will interfere with several components on the bottom of the 41CL board. The spacer and connectors should look like those in the photograph below for a proper installation.

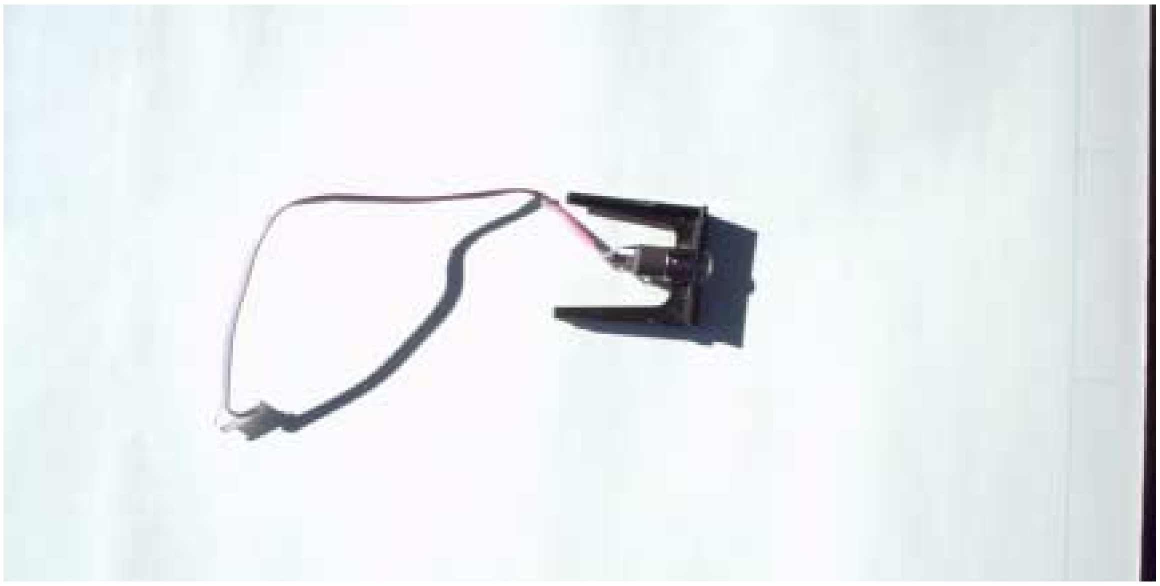


10. Using only the edges of the 41CL circuit board place this board on the main circuit board, with the two lower screw posts going through the holes in the 41CL circuit board just as they did with the original CPU circuit board. Use the antistatic bag containing the 41CL circuit board to store the old CPU circuit board. Do not throw away the old CPU circuit board, as it may still have utility in the future. If you don't want to store the old CPU circuit board, send it to us. If you are installing the optional serial connector, proceed with the steps below. Otherwise skip ahead to step 15.

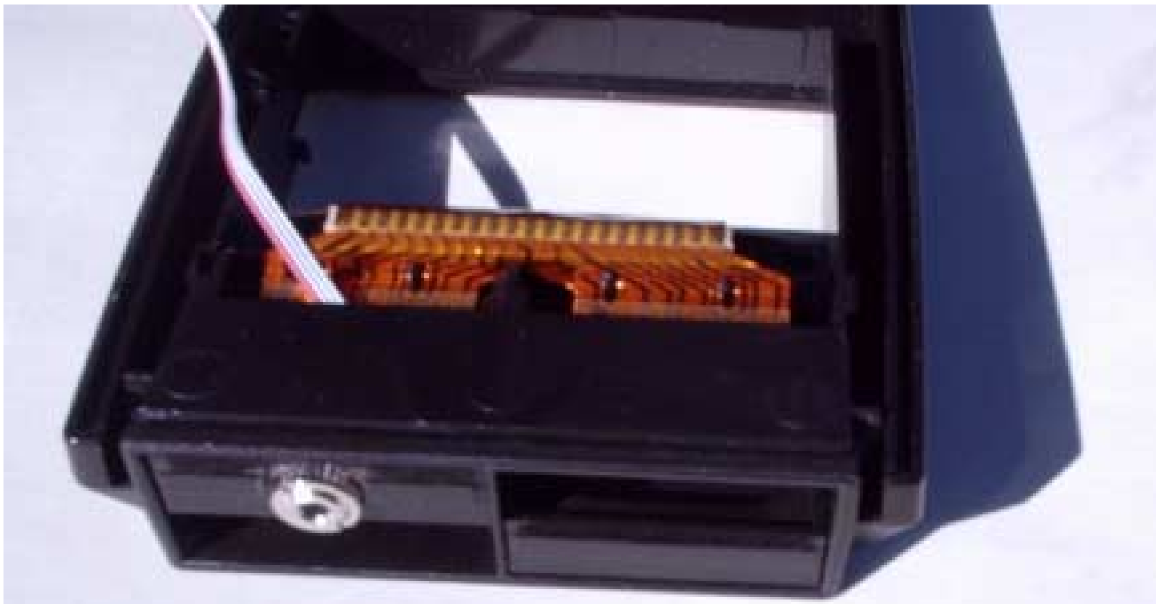


11. The 41CL circuit board contains a simple RS-232C serial port. The Receive Data, Transmit Data and Ground signals for the serial port are present on the programming connector for the CPLD on the board. The optional serial connector contains a plug for this connector, connected through a cable to a 2.5mm stereo jack mounted in a blank Port cover. The serial connector is designed to occupy Port 1 on the calculator only. While it can be plugged into any other Port, doing so will interfere physically with the remaining Ports because of the cable.

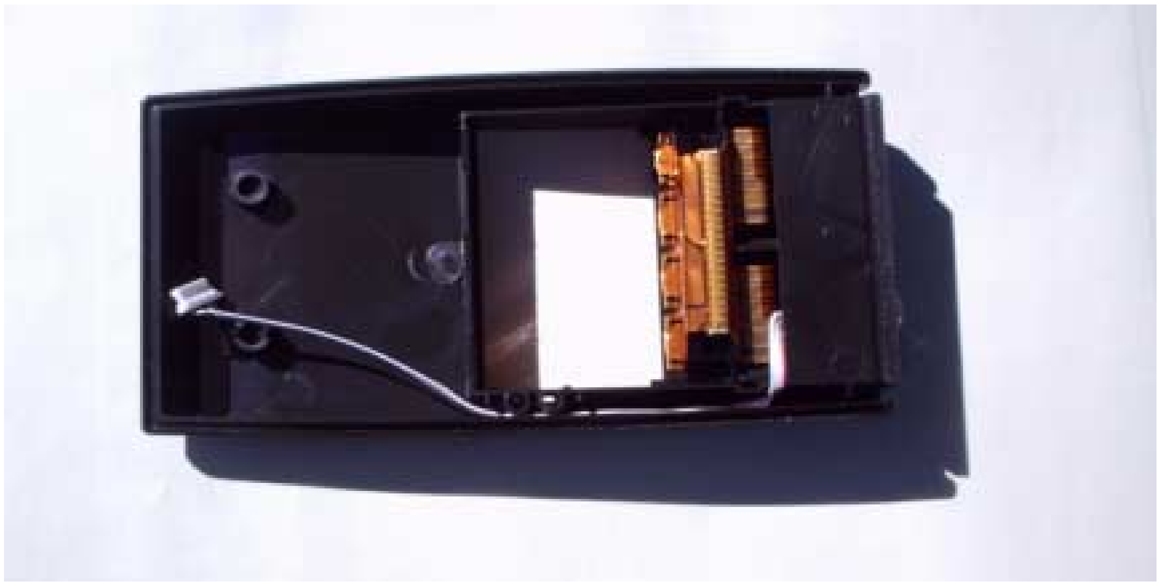
CAUTION! The tension holding a 2.5mm plug in the serial connector jack is higher than the tension holding the blank port cover in the calculator body. This means that trying to pull out the plug will tend to pull the blank port cover out of the calculator, potentially damaging the internal connections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



12. Insert the connector end of the serial connector through Port 1 and in between the space between the calculator body and the flexible Port connectors



13. Route the cable down the side of the calculator body, between the edge of the battery compartment and the outside of the case. It is helpful to use double-sided tape to hold the cable in place next to the battery compartment between the Port and the battery charger Port.

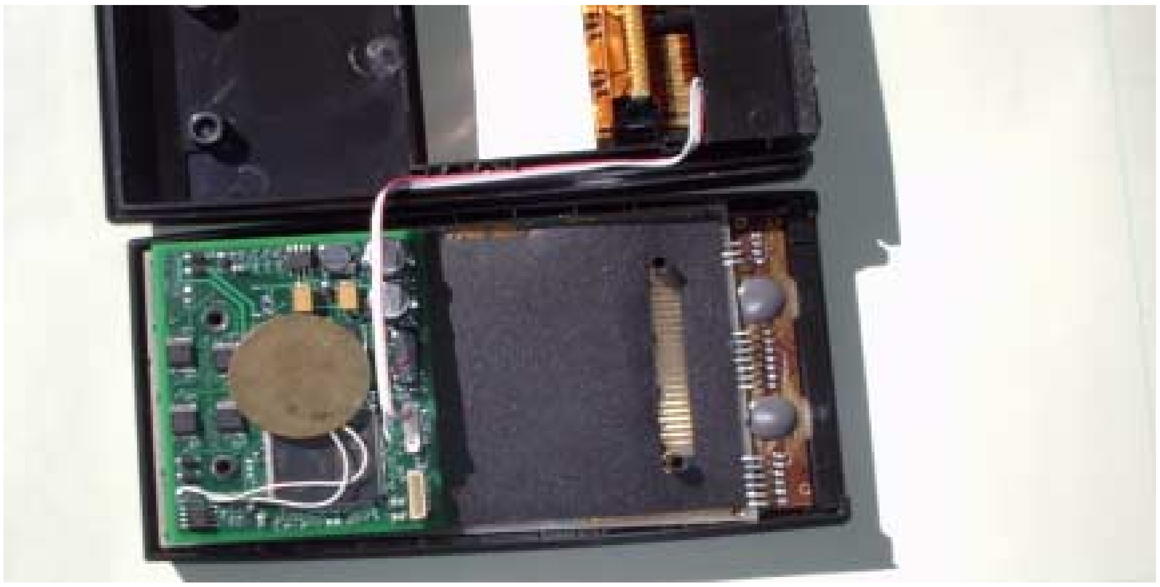


Note how the cable turns sharply near the bottom of the Port to take advantage of the space that will exist between the upper and lower halves of the case. It is easiest to route the cable with the center section of the case in place on the lower case half.

14. Carefully plug the connector on the cable into the connector jack on the 41CL circuit board with the “CP” label next to it. This is the programming connector for the CPLD. The connector used is very fragile and was not really designed for multiple insertions, so take your time, but make sure the plug is fully seated in the connector on the 41CL circuit board.

The two connectors for programming the CPLD and the FPGA on the board are identical, and plugging the serial port connector into the wrong one will damage the board. The 41CL circuit board is shipped with a blank plug in the FPGA connector to prevent accidently inserting the serial connector in the wrong jack.

This step is where the double-sided tape holding the cable in place is useful, as it keeps the cable from popping out of the channel on the side of the calculator where it resides.



15. Carefully fit the center case section (with the proper orientation) and bottom case back together with the remainder of the calculator body. If you installed the serial connector be very careful not to pinch the serial cable between the case halves.

16. Re-install the four screws. The proper way to do this is to slowly turn the screw backwards until you feel the screw threads "click" into the threads in the post. Carefully tighten the screws. **Do not over-tighten the screws** as you risk cracking or shearing off the screw posts. It is best to hold the case sections tightly together with one hand while tightening the screws with the other hand, as this reduces the stress the screws place on the screw posts and case pieces.

17. Re-install the battery case (with the new batteries) and turn the calculator on. If there is no response the flexible connectors are not completely connecting the 41CL circuit board to the main circuit board, and you will need to either tighten the screws a little or reform the circular connector slightly. If garbage rather than the ***MEMORY LOST*** message appears, try removing the battery pack for 30 seconds and re-inserting it again. The display controller contains its own power-on-reset circuit, which sometimes does not properly synchronize with the CPU, leading to garbage in the display on power-up. This occasionally occurs even in the original design. In some cases it may be necessary to do this several times, with a longer time between battery pack insertions.

18. Once the 41CL circuit board installation is verified working, reinstall the calculator feet, and proceed to the initial configuration of the software.

Initial Software Configuration

The 41CL includes a set of functions that provide access to the new features of the NEWT microprocessor. When power is first applied or when the calculator is reset, resulting in the ***MEMORY LOST*** message, the *41CL Extra Functions* are mapped to Page 7 to allow you to do the initial configuration of the calculator. This Page 7 mapping is enforced by the hardware for as long as the Memory Management Unit (MMU) is disabled.

During the initial configuration the *41CL Extra Functions* must either be moved elsewhere so that Page 7 can be used by HP-IL Peripherals or the Page 7 entries in the MMU must be programmed to point to the *41CL Extra Functions* after the MMU is enabled. Until the *41CL Extra Functions* are moved from Page 7 HP-IL will not be available, and an HP-IL Module should not be inserted into the calculator because this will lead to bus conflicts.

There are two images of the *41CL Extra Functions* available, to prevent potential XROM conflicts. The default (YFNZ mnemonic) version uses XROM #15, while the second (YFNS mnemonic) version uses XROM #31. The two copies are identical except for the XROM. Note that some third-party software (CL Utilities, for example) requires the use of the XROM #15 version of the *41CL Extra Functions*.

The copy of the *41CL Extra Functions* that uses the YFNZ mnemonic resides in an area of Flash memory that is protected from modification, while the copy that uses the YFNS mnemonic does not. This was done to provide a way for users to patch the *41CL Extra Functions* in the field if necessary.

Only a few modules use XROM #15, so most users will not need to use the alternate (YFNS) version. There is also a copy of the *41CL Extra Functions Plus* (YFNP mnemonic) available. The *41CL Extra Functions Plus* replaces some of the little-used functions with new functions that allow the user to work with the Image Database (see the 41CL Image Database chapter). As you become more familiar with the 41CL, you may prefer to use the *41CL Extreme Functions* (YFNX mnemonic), which provides a more convenient user interface that prompts for user input.

The minimum sequence for the initial configuration uses three *41CL Extra Functions* (these functions are explained in the next section). If you don’t need to use the advanced features of the 41CL this is sufficient for the initial configuration. This sequence is:

1. **XEQ ALPHA MMUCLR ALPHA** initializes all of the MMU entries in memory, making it safe to enable the MMU in the third step.

2. **ALPHA YFNZ ALPHA XEQ ALPHA PLUG1L ALPHA** plugs the XROM #15 version of *41CL Extra Functions* into the lower half of Port 1 (which is Page 8). Since the MMU is still disabled this has no effect yet. Note that any port can be used for the

41CL Extra Functions. Plugging the *41CL Extra Functions* into a Port allows the use of HP-IL.

Alternatively, **ALPHA YFNP ALPHA XEQ ALPHA PLUG1L ALPHA** plugs the *41CL Extra Functions Plus* into Page 8.

Or you can use **ALPHA YFNS ALPHA XEQ ALPHA PLUGH ALPHA** to plug the XROM #31 version of *41CL Extra Functions* into Page 7. This saves half of a Port, but means that HP-IL (including an HP-IL printer) will not be available. Since the MMU is still disabled this has no effect yet.

3. Finally, **XEQ ALPHA MMUEN ALPHA** enables the MMU, which starts the redirection of either Page 8 or Page 7 to the *41CL Extra Functions*. When using Page 8 any ROM module plugged into Port 1 will not be seen by the 41CL. However, Port 1 can still be used for modules with a fixed address such as the 82143A Printer, the 82160A HP-IL Module, the 82182A Time Module or the 82242A IR Printer Module.

Enjoy your new 41CL calculator! The only thing to remember is that the *41CL Extra Functions* (or the *41CL Extra Functions Plus* or the *41CL Extreme Functions*) **must** remain plugged into a Port for the new features to be available.

If you inadvertently plug another module image into the Port used by the *41CL Extra Functions* or *41CL Extra Functions Plus*, taking the place of these functions as far as the OS is concerned, the only way to recover is either via **BACKSPACE-ON** (causing a ***MEMORY LOST*** condition) or by momentarily removing the battery pack.

The *41CL Extreme Functions* allow the user to protect the MMU programming against accidental modification, and automatically protects itself against accidental deletion.

41CL Extra Functions

The 41CL Extra Functions are required to provide access to the new features of the NEWT microprocessor that powers the 41CL calculator. Depending on your level of experience and how adventurous you are, you may not need all of these functions.

The majority of users will only need the MMU Functions, the Turbo Functions, and the functions that allow you to Plug and Unplug module images to and from the calculator Ports.

If you are a HEPAX user you will need the Memory Block Functions and the Memory/IO Read and Write Functions. Refer to the Using HEPAX section for the details of how to do the initial setup of HEPAX memory.

Users interested in MCODE programming or building a custom module image may find the Memory Buffer Functions useful.

The Image Database functions allow users to customize the Image Database, by adding new mnemonics or modifying existing mnemonics. The Image Database can also be searched using these functions.

The Flash Memory Functions are included for those users wishing to program the Flash memory on the 41CL circuit board beyond the initial programming. Flash memory is non-volatile, so it provides a convenient way to store information permanently. Special areas in Flash memory are reserved for user-programmed module images.

The Serial Port Functions allow users to control the serial port on the 41CL circuit board. Using the serial port will require installation of the optional serial port connector.

Advanced users interested in modifying or replacing the Operating System will need to use the Special MMU Functions. These functions control MMU operation for the pages of memory that contain the Operating System.

The Memory Verification function can be used to verify the integrity of an image in Flash memory or RAM.

Extra Functions Parameter Passing

Many 41CL Extra Functions require hexadecimal (hex) values as arguments, and the ALPHA register is used to hold these arguments. Valid hex digits are the numerals ***0*** - ***9*** and the letters ***A*** - ***F***. Any other character entered as a hex digit will result in a ***DATA ERROR*** message when the function is executed. Any leading zeros must be present for hex numbers.

Multiple hex arguments are separated by either the "***-***" character or the "***>***" character, and these delimiters must be present in the proper location or a ***DATA ERROR*** message will result.

The figure below shows the formatting required when both an address and data are required by the function. All of the ***B***, ***Dx***, ***Lx***, ***Px*** and ***R*** characters are hex digits, and the number and position of the "***-***" characters indicate the type of address.

	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***

A physical address ***P5*** - ***P0*** is a direct physical memory address.

A logical address ***L3*** - ***L0*** is an address used in mcode, where ***L3*** is the address of a Page, and ***L2*** - ***L0*** is the offset within the Page. The bank identifier ***B*** allows the bank to be directly specified (***1***, ***2***, ***3*** or ***4***), for those pages that support multiple banks. In addition, for some functions, all banks (***A***) may be specified. For pages that do not support banks any of these options may be specified, because the field will be ignored. This means that only Bank 1 of a plug-in module can be accessed.

A port address ***R*** is the address of one of the I/O ports contained in the NEWT micropro-cessor. Refer to the Newt Microprocessor Technical Manual for a list of I/O ports and how to use them.

The data field ***D3*** - ***D0*** is a 16-bit data value.

MMU Functions

The MMU Functions allow the user to initialize, enable, disable, and test the status of the MMU.

MMUCLR

Executing **MMUCLR** (*MMU clear*) clears the contents of all of the regular MMU registers (for Pages 4-F) in memory, by writing 0x0000 to these registers. This function should only be executed while the MMU is disabled, to prevent unpredictable results. **MMUCLR** will result in all pages (except for the Operating System, Extended Functions, Time Functions and 41CL Extra Functions) being fetched from the Ports rather than fetched from internal memory.

MMUDIS

Executing **MMUDIS** (*MMU disable*) clears the global MMU enable bit inside the NEWT microprocessor. This automatically reassigns the 41CL Extra Functions to Page 7, but only after the code returns to executing in one of the Operating System pages. This delayed switch allows the function to complete normally and return to the Operating System. This function does not affect the MMU register contents in memory.

MMUEN

Executing **MMUEN** (*MMU enable*) sets the global MMU enable bit inside the NEWT microprocessor. This automatically disables mapping of the 41CL Extra Functions to Page 7, but only after the code returns to executing in one of the Operating System pages. This delayed switch allows the function to complete normally and return to the Operating System. The 41CL Extra Functions must have been assigned to some other page prior to executing **MMUEN**, or the 41CL Extra Functions will no longer be available to the user.

MMU?

Executing **MMU?** (*Test MMU enable*) tests the state of the global MMU enable bit inside the NEWT microprocessor, 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.

Turbo Functions

The Turbo Functions give you control over the operating 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. The current Turbo mode is preserved during idle time, as well as when the calculator is turned off.

Like the Turbo mode performance, the current consumption as a result of Turbo mode is not linear. This is because only a fraction of the circuitry actually runs at a different speed during Turbo mode, in addition to the aforementioned special cases.

TURBOX

Executing **TURBOX** (*Turbo mode disable*) immediately disables any Turbo mode in effect. Any Turbo mode increases power consumption somewhat, so the Turbo modes should be used judiciously if battery life is very important.

TURBO2
TURBO5
TURBO10
TURBO20
TURBO50

Executing **TURBO2** (*Turbo 2X mode*) immediately enables the 2X Turbo mode.

Executing **TURBO5** (*Turbo 5X mode*) immediately enables the 5X Turbo mode.

Executing **TURBO10** (*Turbo 10X mode*) immediately enables the 10X Turbo mode.

Executing **TURBO20** (*Turbo 20X mode*) immediately enables the 20X Turbo mode.

Executing **TURBO50** (*Turbo 50X mode*) immediately enables the 50X Turbo mode.

TURBO?

Executing **TURBO?** (*Test Turbo mode*) queries the state of the Turbo control bits inside the NEWT microprocessor. 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.

Plug into Port/Unplug from Port Functions

These functions allow the user to virtually plug and unplug module images from the calculator ports. The functions use the Image Database in Flash memory to determine the type and address of the desired module image. This provides an easy way for the user to control the configuration of the calculator without having to remember specific memory addresses. Refer to the Image Identifier Table section for the mnemonics for the different module images. No attempt will be made here to explain what the different module images do; it is assumed that the user has access to the documentation for any modules of interest.

Advanced users may enjoy exploring some of the poorly-documented images that are included in the 41CL. Advanced users can even build a new module image in memory and then virtually plug it into a port by specifying the relevant memory address for the Plug function. Or an entry can be added to the Image Database to allow the use of a new mnemonic.

Users should avoid XROM conflicts when using module functions in programs. Refer to the original HP documentation for details. Users can circumvent conflicts by copying a module image to RAM and then manually modifying the XROM number before plugging this modified image into a Port.

Users 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

Note that some third-party modules can be addressed independently of their physical location, but the User must still avoid address conflicts with these modules.

PLUG1 **(image identifier in ALPHA register)**
PLUG2
PLUG3
PLUG4

Executing **PLUG1** (*Plug into Port 1*) inserts a module image into Port 1, which is Pages 8 and 9 of the logical address space. This function automatically programs the MMU registers for both Pages 8 and 9 (all banks) as appropriate for the selected module image. Module images that are only one page long will be loaded into the lower page and the upper page will be left empty.

The four-character module identifier must be properly formatted in the ALPHA register or a ***BAD ID*** message will result.

If the module image cannot be used in Pages 8 and 9 a ***DATA ERROR*** message will result.

If the Image Database cannot be found in Flash memory a ***NO IMDB*** message will result. If the corresponding entry in the Image Database has not been programmed a ***NO ENTRY*** message will be returned, and if the entry has been zeroed out (deleted) a ***NULL ENTRY*** message will be returned.

The **PLUG2** (*Plug into Port 2*) function operates on Port 2 (Pages A and B).

The **PLUG3** (*Plug into Port 3*) function operates on Port 3 (Pages C and D).

The **PLUG4** (*Plug into Port 4*) function operates on Port 4 (Pages E and F).

PLUG1L **(image identifier in ALPHA register)**
PLUG2L
PLUG3L
PLUG4L
PLUGP

The **PLUG1L** (*Plug into Port 1 lower half*) function is identical to the **PLUG1** function except that it only operates on the lower half of Port 1 (Page 8 of the logical address space). This function can only be used with module images that are one page long.

The **PLUG2L** (*Plug into Port 2 lower half*) function operates on the lower half of Port 2 (Page A).

The **PLUG3L** (*Plug into Port 3 lower half*) function operates on the lower half of Port 3 (Page C).

The **PLUG4L** (*Plug into Port 4 lower half*) function operates on the lower half of Port 4 (Page E).

The **PLUGP** (*Plug into printer page*) function operates on the page normally used for a printer (Page 6). Do not plug a printer into the calculator if you have virtually plugged an image into the Page 6. Page 6 is special in that the Operating System often calls printer routines in this page. Normally, if no printer is plugged into the calculator the software immediately returns to the Operating System, without ill effect. So any image plugged into Page 6 with the **PLUGP** function must keep these printer subroutine entry points clear so that the Operating System can function properly. Only these images are known to be compatible with Page 6:

Page 6 compatible images	Mnemonic
HEPAX	HEPX or HPX2
Power CL Utilities	PWRL

PLUG1U **(image identifier in ALPHA register)**
PLUG2U
PLUG3U
PLUG4U
PLUGH

The **PLUG1U** (*Plug into Port 1 upper half*) function is also identical to the **PLUG1** function except that it only operates on the upper half of Port 1 (Page 9 of the logical address space). This function can only be used with module images that are one page long.

The **PLUG2U** (*Plug into Port 2 upper half*) function operates on the upper half of Port 2 (Page B).

The **PLUG3U** (*Plug into Port 3 upper half*) function operates on the upper half of Port 3 (Page D).

The **PLUG4U** (*Plug into Port 4 upper half*) function operates on the upper half of Port 4 (Page F).

The **PLUGH** (*Plug into hp-il page*) function operates on the page normally used for the HP-IL module (Page 7). Do not plug an HP-IL module into the calculator if you have virtually plugged an image into the page 7.

The various **PLUG** functions can also be used to assign pages in physical memory directly to the Ports. The upper three nibbles of the physical memory address (so that it begins on a 4K boundary) are specified, and the **PLUG** function assigns either one page independent of bank; two pages independent of bank; one page with all four banks; or two pages, each with all four banks.

	ALPHA register							
	7	6	5	4	3	2	1	image characteristics
format	***P5***	***P4***	***P3***	**-**	***1***	***6***	***K***	16K (one page, four banks)
format	***P5***	***P4***	***P3***	**-**	***D***	***B***	***L***	8K (two pages)
format	***P5***	***P4***	***P3***	**-**	***R***	***A***	***M***	4K (one page)
format	***P5***	***P4***	***P3***	**-**	***M***	***A***	***X***	32K (two pages, each with four banks)

The ***-DBL***, ***-RAM***, ***-16K*** and ***-MAX*** mnemonics can be used with either RAM or Flash memory addresses. These mnemonics are decoded independent of the Image Database, so that they can be used even if the Image Database is not present in memory. The same is true for the ***YFNP***, ***YFNS*** and ***YFNZ*** mnemonics.

UPLUG1
UPLUG2
UPLUG3
UPLUG4

Executing **UPLUG1** (*Unplug from Port 1*) removes the module image from Port 1, which is Pages 8 and 9 of the logical address space. The function does this by clearing the MMU entries for these pages.

The **UPLUG2** (*Unplug from Port 2*) function operates on Port 2 (Pages A and B).

The **UPLUG3** (*Unplug from Port 3*) function operates on Port 3 (Pages C and D).

The **UPLUG4** (*Unplug from Port 4*) function operates on Port 4 (Pages E and F).

UPLUG1L
UPLUG2L
UPLUG3L
UPLUG4L
UPLUGP

The **UPLUG1L** (*Unplug from Port 1 lower half*) function is identical to the **UPLUG1** function except that it only operates on the lower half of Port 1 (Page 8 of the logical address space). Be careful using this function if you are planning on plugging a physical module into Port 1, because the upper half of the Port will still be fetched from internal memory.

The **UPLUG2L** (*Unplug from Port 2 lower half*) function operates on the lower half of Port 2 (Page A).

The **UPLUG3L** (*Unplug from Port 3 lower half*) function operates on the lower half of Port 3 (Page C).

The **UPLUG4L** (*Unplug from Port 4 lower half*) function operates on the lower half of Port 4 (Page E).

The **UPLUGP** (*Unplug from printer page*) function operates on Page 6.

UPLUG1U
UPLUG2U
UPLUG3U
UPLUG4U
UPLUGH

The **UPLUG1U** (*Unplug from Port 1 upper half*) function is identical to the **UPLUG1** function except that it only operates on the upper half of Port 1 (Page 9 of the logical address space). Be careful using this function if you are planning on plugging a physical module into Port 1, because the lower half of the Port will still be fetched from internal memory.

The **UPLUG2U** (*Unplug from Port 2 upper half*) function operates on the upper half of Port 2 (Page B).

The **UPLUG3U** (*Unplug from Port 3 upper half*) function operates on the upper half of Port 3 (Page D).

The **UPLUG4U** (*Unplug from Port 4 upper half*) function operates on the upper half of Port 4 (Page F).

The **UPLUGH** (*Unplug from hp-il page*) function operates on Page 7.

Memory Block Functions

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

Given that they are operating on 4096 memory locations, the 41CL is automatically switched to the 50x Turbo mode during the transfer. The current Turbo mode is restored after the transfer is complete.

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. Only memory addresses are valid for this function, and a ***DATA ERROR*** message will result if an I/O address is specified. 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 RAM memory. This function only allows copying blocks of memory that start on 4K boundaries and are 4K in length. Only memory addresses are valid for this function, and ***DATA ERROR*** will result if an I/O address is specified.

When executed from the keyboard a ***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 the 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 register memory, program the MMU, update the register address information 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 NEWT 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 NEWT 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 NEWT 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 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 4K block (one page) of System memory to be used as a buffer for assembling module images. The Extra Functions Buffer Area is located at physical addresses 0x805000 - 0x805FFF, and has an associated Extra Functions Buffer Pointer stored at address 0x804010. The memory buffer functions provide a convenient way to move data to the buffer 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 Buffer Pointer, which is automatically incremented by the **YBUILD** function after use.

Thus, to assemble blocks of code the user merely initializes the Buffer Pointer to the start of the block, with either 0x000 if assembling a FAT, or 0x084 if assembling functions, and then copies 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 for use.

Of course all of the normal memory functions may be used with the Buffer Area, and indeed the region can also be used as normal memory when not being used as the buffer.

The Memory Buffer functions are not present in the 41CL Extra Functions Plus, having been replace by the Image Database functions in that image.

YBPNT (data in ALPHA register)

Executing **YBPNT** (*Write Extra Functions buffer pointer*) writes data directly to the Extra Function Buffer Pointer at address 0x804010.

The data must be a four-digit hex number but only the lower three digits of this value are used. The most-significant digit is ignored and not changed by the buffer functions.

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

The function returns 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
physical address	***8***	***0***	***4***	***0***	***1***	***0***	***-***	***D3***	***D2***	***D1***	***D0***

YBPNT?

Executing **YBPNT?** (*Read Extra Functions buffer pointer*) reads directly from Extra Function Buffer Pointer at address 0x804010. The function returns 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
physical address	***8***	***0***	***4***	***0***	***1***	***0***	***-***	***D3***	***D2***	***D1***	***D0***

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 Extra Functions 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 ***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.

Care must be exercised 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.

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.

	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***

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 accesses of the Flash memory are allowed. This means that these functions must be running out of RAM to work. Both Flash Memory functions check for this, and return with a ***CODE=ROM*** error message if this is not the case. If you really want to use either of the Flash Memory functions you must copy the entire 41CL Extra Functions image to RAM and then program the MMU to use this RAM copy of these functions

These functions cannot be used to modify to the Operating System area (the first sector in the Flash) and will return with the ***OS AREA*** error message if an address in the first sector of the Flash memory is specified as the destination.

YFERASE (address in ALPHA register)

Executing **YFERASE** (*Erase Flash 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. An address that is in RAM (***P5*** is 8 or greater) will return immediately with the ***DST=RAM*** error message.

	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 page) copies the contents of one 4K block (one page) of RAM memory to a 4K block of Flash memory. This function only allows copying block 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. A destination address that is in RAM (destination ***P5*** is 8 or greater) will return with the ***DST=RAM*** error message, and a source address that is in Flash (source ***P5*** is 7 or less) will return with the ***SRC=ROM*** message.

	ALPHA register						
	7	6	5	4	3	2	1
		source				destination	
physical address to physical address	***P5***	***P4***	***P3***	**>**	***P5***	***P4***	***P3***

Serial Port Functions

The 41CL contains an RS-232 serial port, but unless you have the special PCB connector this functionality will not be available. However, these serial port functions are present on all 41CL circuit boards.

The serial port hardware is initialized and the baud rate is set to 1200 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 may be 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 serial 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.**

SERINI

Executing **SERINI** (*Initialize serial port*) initializes the serial port 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 has no effect on the RS-232 driver.

BAUD12
BAUD24
BAUD48
BAUD96

Executing **BAUD12** (*Select 1200 baud*) sets the baud rate for the serial port to 1200. This is the default selection for the serial port, and is automatically selected when the calculator is turned on or when the **SERINI** function is executed.

Executing **BAUD24** (*Select 2400 baud*) sets the baud rate for the serial port to 2400.

Executing **BAUD48** (*Select 4800 baud*) sets the baud rate for the serial port to 4800.

Executing **BAUD96** (*Select 9600 baud*) sets the baud rate for the serial port to 9600.

YGETLB
YGETUB
(address in the ALPHA register)

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 as the address.

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 as the address.

These functions do not check the address except for proper formatting, so attempting to write to Flash memory is allowed, although it will be ignored by the Flash memory.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YGETLB** and **YGETUB** functions. These functions put a ***RECEIVING*** message in the display while waiting for a character, and will return with a ***TIMEOUT*** error message after the time-out period if no receive byte is available. 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. In this case the function will return with an ***OVERRUN*** error message.

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

YPUTLB
YPUTUB
(address in the ALPHA register)

Executing **YPUTLB** (*Write lower memory byte to serial por*t) reads the lower byte from the specified address and attempts to write it to the serial port.

Executing **YPUTUB** (*Write upper memory byte to serial por*t) reads the upper byte from the specified address and attempts to write it to the serial port.

The figure below shows the formatting required for the address and data in the ALPHA register for the **YPUTLB** and **YPUTUB** functions. These functions put a ***SENDING*** message in the display while waiting to send a character, and will return with a ***TIME-OUT*** error message after the time-out period if the transmitter cannot accept a byte.

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

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 ***SENDING*** message in the display while waiting to send characters, and will return with a ***TIMEOUT*** error message after the time-out period if the transmitter cannot accept a byte at any point during the block transfer.

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 ***RECEIVING*** message in the display while waiting for a character, and will return with a ***TIMEOUT*** error message after the time-out period if no receive byte is available at any point during the block transfer. 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. In this case the function will return with an ***OVERRUN*** error message.

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.

Miscellaneous Functions

YFNS?

Executing **YFNS?** (*Read 41CL Extra Functions location*) polls the logical memory for the current location of the 41CL Extra Functions. The page where the 41CL Extra Functions reside is returned in the X register as a decimal number in the range ***6*** through ***15***, corresponding to Pages 6 through F. This information can be used to prevent accidentally overwriting the 41CL Extras Functions when reprogramming the MMU.

YCRC (Page address in ALPHA register)

Executing **YCRC** (*Calculate CRC on page*) calculates the 32-bit Cyclic Redundancy Check (CRC) used for Ethernet over the 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
physical page address	***P5***	***P4***	***P3***

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 ***WORKING*** message in the display while running.

Image Database Functions

The Image Database Functions allow the user to add to, modify, store, and search 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, located at physical address 0x805000. 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.

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

Executing **IMDB?** (*Search Image Database*) 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 tests that the Image Database is present, and returns with a ***NO IMDB*** error message if this is not true.

The figure below shows the formatting for a module identifier and 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 below. 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. If no address match is found the function will result in a ***NO MATCH*** error message. Only the first match (the search proceeds from lowest database address upwards) will ever be returned.

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

Searches with a valid module identifier always return the corresponding contents of the Image Database, even if the entry is unprogrammed.

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)

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 ***SEARCHING*** message is written to the display while a search is in progress.

The normal 41CL Extra Functions includes the **IMDB?** function, but in this case the function always searches the Flash version of the Image Database.

IMDBF

Executing **IMDBF** (*Image Database in Flash*) 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?** or **IMDBINS** functions. The selection information is stored in RAM at location 0x804014, in the least-significant digit.

This function is only present in the 41CL Extra Functions Plus.

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.

This function is only present in the 41CL Extra Functions Plus.

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.

This function is only present in the 41CL Extra Functions Plus.

IMDBCPY

Executing **IMDBCPY** (*Copy Image Database to RAM*) copies the Image Database at address 0x0DF000 in Flash memory to the Extra Functions Buffer area at address 0x805000.

This function tests that the Image Database is present at address 0x0DF000, and returns with a ***NO IMDB*** error message if this is not true. This function automatically executes in 50x Turbo mode and is equivalent to **YMCPY** with ***0DF>805*** in the ALPHA register.

This function is only present in the 41CL Extra Functions Plus.

IMDBUPD

Executing **IMDBUPD** (*Update Image Database in Flash*) writes the contents of the Extra Functions Buffer area at address 0x805000 to Flash memory starting at address 0x0DF000.

This function tests that the Image Database is present at address 0x805000, and returns with a ***NO IMDB*** error message if this is not true. This function checks that it is executing from RAM, and returns with a ***CODE=ROM*** error message if this is not the case. **IMDBUPD** 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. This function tests that the Image Database is present, and returns with a ***NO IMDB*** error message if this is not true.

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***

When using the copy of the Image Database in RAM the new entry is unconditionally written to the appropriate location.

When using the copy of the Image Database in Flash memory the function first checks that it is executing from RAM, and returns with a ***CODE=ROM*** error message if this is not the case. 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.

Special MMU Functions

The Special MMU Functions allow the user to enable and disable the MMU translation circuitry for the Operating System pages (0-3 and 5). Normally these pages are never mapped by the MMU, for obvious reasons. But users wanting to experiment with an alternative Operating System can use the **MAPEN** command to enable the special MMU translation circuitry.

If the new Operating System uses the same register assignments and calling conventions, then the **MAPDIS** command can be used from the modified Operating System to revert to the native Operating System.

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.

The MMU storage locations for the Operating System pages (Pages 0-3) do not follow the convention used for all other memory pages, and these pages do not support multiple banks. The table below shows the MMU storage locations for Pages 0-3.

Physical Address	Contents
0x80400C	MMU register for Page 0
0x80401C	MMU register for Page 1
0x80402C	MMU register for Page 2
0x80403C	MMU register for Page 3

All of the MMU storage locations for Pages 0-3 must be initialized before executing the **MAPEN** function, because these registers are not initialized by the **MMUCLR** function. This will preserve the MMU mapping for the Operating System when the calculator is turned off, so that only the global enable needs to be set to restore the mapping function. The individual enable bits for each page still apply, making it easy to map just part of the Operating System.

MAPDIS

Executing **MAPDIS** (*Disable special MMU mapping*) clears the special MMU enable bit inside the NEWT microprocessor, 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, in case the modified Operating System uses a different convention.

MAPEN (passphrase in ALPHA register)

Executing **MAPEN** (*Enable special MMU mapping*) sets the special MMU enable bit inside the NEWT microprocessor, but only if the correct passphrase (***NEW OS***) is present in the ALPHA register.

An incorrect passphrase will result in a ***PASS ERR*** message.

This function automatically enables mapping of the Operating System pages, but only if the MMU is globally enabled. The **MAPEN** and **MMUEN** commands may be issued in either order. All of the normal MMU entries must be valid before the **MMUEN** command is issued, and all of the special MMU entries (Pages 0-3 and 5) must be valid before the **MAPEN** command is issued.

The **MAPEN** function uses the normal 41C call/return convention, which means that the function will return through address 0x00F0. Any modified Operating System must take this into account. If necessary, this function can be patched to return through address 0x0000. Contact the factory for the details of this patch.

Error Messages

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

Error Message	Function	Meaning
ADDR ERROR	**YFERASE** **YFWR**	Address is outside of Flash address range (only in versions subsequent to -4D)
BAD ID	**IMDB?** **IMDBINS** **PLUG**	Invalid module ID in ALPHA
CODE=ROM	**IMDBINS** **IMDBUPD** **YFERASE** **YFWR**	Trying to execute function from Flash (only in versions prior to -4E)
DATA ERROR	**IMDB?** **IMDBINS** **PLUG** **YBPNT** **YBUILD** **YCRC** **YEXP** **YFERASE** **YFWR** **YGET** **YIMP** **YMCLR** **YMCPY** **YPOKE** **YPEEK** **YPUT**	Invalid hexadecimal in ALPHA

DST=RAM	**YFERASE** **YFWR**	Attempting Flash operation on RAM (only in versions prior to -4E)
NO ENTRY	**PLUG**	unprogrammed entry in Image Database
NO IMDB	**IMDB?** **IMDBF?** **IMDBCPY** **IMDBINS** **IMDBUPD** **PLUG**	No Image Database found
NO MATCH	**IMDB?**	No address match found in Image Database
NULL ENTRY	**PLUG**	empty (all zeros) entry in Image Database
OS AREA	**YFERASE** **YFWR**	Attempting Flash operation on Operating System
OVERRUN	**YGET** **YIMP**	Receiver overrun detected
PASS ERR	**MAPEN**	Incorrect passphrase in ALPHA
SRC=ROM	**YFWR**	Trying to transfer from Flash to Flash
TIMEOUT	**YEXP** **YGET** **YIMP** **YPUT**	Timeout during attempted transfer
TYPE ERR	**PLUG**	Unknown image type

Function Summary

The table below lists all of the *41CL Extra Functions* (and *41CL Extra Functions Plus*), along with the arguments and return values.

Function	Arguments (ALPHA)	Returns (X)	Returns (ALPHA)	Returns (Display)	Def	Plus	Notes
BAUD12 **BAUD24** **BAUD48** **BAUD96**					x	x	
IMDB?	module ID or address		IMDB entry	IMDB entry	x	x	Module ID query always returns IMDB entry
IMDBF						x	
IMDBF?						x	
IMDBCPY						x	
IMDBINS	IMDB entry					x	
IMDBR						x	
IMDBUPD						x	
MAPDIS					x	x	
MAPEN	passphrase				x	x	
MMUCLR					x	x	
MMUDIS					x	x	
MMUEN					x	x	

MMU?				***NO*** ***YES***	x	x	MMU is disabled MMU is enabled
PLUG1 **PLUG1L** **PLUG1U** **PLUG2** **PLUG2L** **PLUG2U** **PLUG3** **PLUG3L** **PLUG3U** **PLUG4** **PLUG4L** **PLUG4U** **PLUGH** **PLUGP**	module ID				x	x	
SERINI					x	x	
TURBOX					x	x	
TURBO2 **TURBO5** **TURBO10** **TURBO20** **TURBO50**					x	x	
TURBO?		***0*** ***2*** ***5*** ***10*** ***20*** ***50***			x	x	Turbo mode disabled 2x Turbo mode 5x Turbo mode 10x Turbo mode 20x Turbo mode 50x Turbo mode

UPLUG1 UPLUG1L UPLUG1U UPLUG2 UPLUG2L UPLUG2U UPLUG3 UPLUG3L UPLUG3U UPLUG4 UPLUG4L UPLUG4U UPLUGH UPLUGP					x	x	
YBPNT	data				x		
YBPNT?			address/ data	address/ data	x		Buffer Pointer value is in the data field
YBUILD	address/length				x		
YCRC	address		CRC	CRC		x	
YEXP	address/length				x	x	
YFERASE	address				x	x	
YFNS?		*6-15*			x	x	Logical address page where YFNS currently resides
YFWR	address				x	x	
YGETLB YGETUB	address				x		
YIMP	address/length				x	x	
YMCLR	address/data				x	x	
YMCPY	address pair				x	x	
YPOKE	address/data				x	x	
YPEEK	address/data		address/ data	address/ data	x	x	input data field is replaced with actual data
YPUTLB YPUTUB	address				x		

Image Identifiers

The table below shows the module images that are present in the Flash memory of the 41CL, along with the mnemonics for use with the **PLUG** and **PPLUG** functions, any restrictions on module image placement, the group the image is in, and the XROM numbers used by the image. Four character mnemonics were chosen to allow easy-to-remember ALPHA contents, but only the first and last characters of the mnemonics are parsed by the **PLUG** and **PPLUG** functions.

Mnemonic	Description	Restrictions	Group	XROM
AADV	Advantage Applications ROM		**ENG**	19
ADV1	Adventure ROM, part 1	4 pages	**GAM**	12
ADV2	Adventure ROM, part 2	4 pages	**GAM**	13
AEC3	AECROM III Module	2 pages	**GEN**	18
AECR	AECROM Module	2 pages	**GEN**	18
AFDE	AFDC1 ROM	2 pages	**GOV**	16/17
AFDF	AFDC2 ROM	2 pages	**GOV**	18/19
AFIN	Autofinance Module		**FIN**	21
ALGG	Algebra ROM	2 pages	**MAT**	6/7
ALGY	Astrology ROM		**AST**	31
ALPH	Alpha ROM		**UTL**	6
ANGZ	Angel’s ZEPROM	2 pages	**UTL**	3, 12
ANTS	HP-41 Antennas Solutions		**ENG**	16
AOSX	AMC-OSX ROM		**SYS**	5
ASM4	Assembler 4		**UTL**	21
ASMB	Assembler 3		**UTL**	21
ASTT	Astro-2010 ROM and Astro-2010 UI ROM	4 pages	**AST**	6/8
AUTO	HP Autostart		**HIL**	10
AV1Q	Beechcraft ROM		**AVI**	31
AVIA	HP Aviation Pac 1A		**AVI**	19
B52B	Boeing-B52 Module	2 pages	**GOV**	21/31

BASI	BASIC ROM		**PRG**	8
BBSC	BBS ROM	4 pages	**FIN**	11/12/13/14
BCMW	BCMW ROM		**ENG**	8
BESL	Bessel Functions ROM	2 pages	**MAT**	2/3
BJMX	Blackjack MAX -2E		**GAM**	6
BLDR	BLD ROM		**UTL**	17
BLJK	Blackjack -1B		**GAM**	7
BLND	BufferLand ROM	after Z41Z	**UTL**	8
BSMS	HP-41 Business Sales/Marketing/Stats Solutions		**FIN**	18
BUD2	Buderus-2 Module	2 pages	**ENG**	15/16
BUD3	Buderus-3 Module		**ENG**	9
CCDA	Advanced CCD ROM		**GEN**	10
CCDP	CCD Plus Module	2 pages	**UTL**	9/11
CCDR	CCD Module 1B	2 pages	**UTL**	9/11
CCDX	CCD OS/X		**SYS**	5
CENG	Chemical Engineering Solutions		**ENG**	12
CHEM	Chemistry User Module		**CHM**	20
CHES	Chess ROM	2 pages	**GAM**	8
CIRC	HP Circuit Analysis Pac 1A		**ENG**	6
CIVI	HP-41 Civil Engineering Solutions		**ENG**	16
CIVU	Civil Engineering Special Collection	2 pages	**ENG**	20
CLIN	HP Clinical Lab & Nuclear Medicine Pac 1A		**MED**	19
CLND	HP Calendar Solutions		**GEN**	12
CLUT	CL Utilities ROM		**UTL**	12
CMT1	CMT-100 EPROM Test ROM	4 pages	**HWS**	31
CMT2	CMT-200 Data Acquisition ROM		**HWS**	4
CMT3	CMT-300 Multimeter ROM	2 pages	**HWS**	9
CNTL	HP-41 Control Systems Solutions		**ENG**	14
COOQ	CO-OP ROM	2 pages	**SVY**	31
CRTO	Cryptography Module		**MAT**	10
CURV	CurveFit ROM	2 pages	**MAT**	4/5
CVPK	CVPAK ROM	2 pages	**ENG**	21/31
DA4C	Disasm 4C		**UTL**	15
DACQ	HP Data Acquisition 1B	2 pages	**HWS**	21/31
DASM	Disasm 4D		**UTL**	15
DAVA	David Assembler 2C		**UTL**	2
DEMO	HP-41 System Demo Program	4 pages	**GEN**	14
DEV2	HP-IL Development Pac 2	2 pages	**HIL**	22/24
DEVI	HP HP-IL Development Pac 1B	2 pages	**HIL**	22/24

DIFF	Differential Equations ROM	2 pages	**MAT**	15
DIGT	DigitPAC ROM		**ENG**	24
DIIL	HP HP-IL Diagnostic		**HIL**	19
DMND	Diamond ROM		**FIN**	31
DST1	CalTrans Survey	4 pages	**SVY**	8/9/10
DYRK	Dyerka ROM		**UTL**	31
E41S	ES41 Module	2 pages	**HWS**	4/6
EEFD	EE Filter Design ROM	2 pages	**ENG**	17/18
EENG	HP-41 Electrical Engineering Solutions		**ENG**	15
EILP	Extended IL Plus ROM		**HIL**	27
EPRH	MLEPR -1H ROM		**UTL**	4
EPRM	MMEPROM ROM		**UTL**	16
EPTN	Trans-Neptunian Planets 2016-2025		**AST**	23
ESML	ESMLDL 7B Module		**HWS**	10
ETS3	ETSII3 ROM	2 pages	**ENG**	12
ETS4	ETSII4 ROM	2 pages	**ENG**	8/14
ETS5	ETSII5 ROM	2 pages	**ENG**	10/20
ETS9	ETSII6 ROM		**ENG**	16
EXIO	HP Extended I/O 1A		**HIL**	23
EXTI	SKWID EXT-IL ROM		**HIL**	27
FACC	300889 FACC ROM	2 pages	**GOV**	10/11
FAIR	Fairfield ROM	2 pages	**ENG**	21/31
FCS2	Forecast 2 ROM		**FIN**	10
FCST	Forecast ROM		**FIN**	10
FDYN	Fluid Dynamics Solutions		**ENG**	17
FFEE	For Fee ROM		**ENG**	14
FINA	HP Financial Decisions Pac 1D		**FIN**	4
FLDB	41CL Flash YCRC Database	not pluggable	**OSL**	N/A
FRTH	FORTH ROM	Pages 4 & 7	**PRG**	N/A
FSSY	FOCAL Assembly/Disassembly ROM	2 pages	**PRG**	14
FUNS	Funstuff ROM	4 pages	**GAM**	10
GAME	HP Games Pac 1A		**GAM**	10
GEOM	HP-41 Geometry Solutions		**MAT**	14
GMAS	GMAC 2 Module		**FIN**	31
GMAT	GMAC 3 Module	2 pages	**FIN**	21/31
GMTY	Geometry-11 ROM		**MAT**	16
GRAW	Gene's RAW files	2 pages	**GEN**	18
GRVI	Gravity & Time ROM	2 pages	**PHY**	16
GSLV	Geometric Solver ROM		**MAT**	18

GTWN	Ghost Town ROM		GAM	12
HCMP	Hydracomp ROM		ENG	21
HCPL	Hyper-Complex Math ROM	3 pages	MAT	10/20/21
HDIS	HEPAX Disassembler		UTL	9
HEP2	Modified HEPAX (1E) ROM		SYS	7
HEPR	HEPAX RAM Template		SYS	N/A
HEPX	HEPAX 1D Module		SYS	7
HMAT	HP-41 High-Level Math Solutions		MAT	12
HOME	HP Home Management Pac 1A		FIN	9
HSRV	HP Service Module	Page 4	OSL	N/A
HVAC	HVAC Solutions		ENG	16
IBOX	ICEbox 1H		UTL	4
ICOD	ICODE ROM		PRG	19
IDC1	IDC1 ROM		MED	21
IDC2	IDC2 ROM	2 pages	MED	21/22
IERR	IERR ROM	2 pages	MAT	1
ILBF	IL Buffer ROM		HIL	22
IMDB	41CL Image Database	not pluggable	OSL	N/A
INDO	Philips Indoor Lighting ROM		ENG	10
INTG	Integration ROM	2 pages	MAT	16
ISEN	ISENE ROM		GEN	17
ISOL	Interchangeable Solutions ULPE Program		PRG	11
ITCP	NutIP TCP/IP ROM		HIL	4
JARR	K. Jarrett XF/SP books	2 pages	GEN	17/18
JMAT	JMB-Math ROM	2 pages	MAT	5/6
JMBC	JMB-Calendar ROM		GEN	17
JMTX	JMB-Matrix ROM	2 pages	MAT	8
K135	KC-135 Weight & Balance ROM	4 pages	GOV	16/21/31
KRGM	Kruse/Gosmann books	2 pages	GEN	17/18
L119	AFDC-L119 ROM	2 pages	GOV	21/31
LAIT	Laitram XQ2 ROM	Page 4	SYS	N/A
LAND	LandNav Module		NAV	1
LBLS	Labels ROM		UTL	
LENG	Solar Engineering Solutions		ENG	14
LNDL	HP-41 Lend, Lease Savings Solutions		FIN	19
LPLC	Laplace Transform ROM	2 pages	MAT	10
MADV	Modified Advantage Pac	2 pages	MAT	24/26
MASS	Extended Mass Storage ROM		HIL	16
MATH	HP Math Pac 1D		MAT	1

MBFR	Memory Functions Buffer Area (RAM page 806)	not pluggable	OSL	N/A
MCCK	Alan McCornack book		GEN	16
MCHN	HP Machine Design Pac 1A		ENG	12
MCMP	Mountain Computer 1C		HWS	15
MDP1	MDP1 ROM	2 pages	ENG	15/16
MDP2	MDP2 ROM	2 pages	ENG	17/18
MELB	Melbourne ROM		GEN	12
MENG	HP-41 Mechanical Engineering Solutions		ENG	16
MILE	Military Engineering	2 pages	GOV	21/31
MLBL	David Assembler Labels ROM		UTL	
MLRM	MLROM		UTL	21
MLTI	Multi-Precision Library ROM		MAT	3
MONO	Monopoly ROM	2 pages	GAM	16
MTRA	Advanced Matrix ROM	4 pages	MAT	22/24
MTRX	Matrix ROM		MAT	7
MTST	MCTEST ROM		HWS	3
MUEC	Muecke ROM	2 pages	ENG	21/31
MWK3	MWK-3 Module		ENG	10
MWK4	MWK-4 Module	2 pages	ENG	21/31
NAVI	HP Navigation Pac 1B	2 pages	NAV	14
NCHP	NOV CHAP ROM		UTL	31
NEA1	SNEAP-1 and -2 ROM	2 pages	ENG	21/31
NEA3	SNEAP-3 and -4 ROM	2 pages	ENG	11/10
NEA5	SNEAP-5 and -6 ROM	2 pages	ENG	13/14
NEXT	NEXT ROM		UTL	6
NFCR	NFCROM 1B		HWS	17
NONL	Non-linear Systems Module		MAT	16
NPAC	Navpac ROM	2 pages	NAV	14/15
NTHY	Number Theory ROM		MAT	16
NVCM	NAVCOM 2	2 pages	NAV	14/15
OBCZ	OBCSYS ROM		MED	31
OILW	Oilwell ROM	2 pages	ENG	21/31
OPTO	HP-41 Optometry Solutions		MED	16
OS41	HP-41 Operating System	Page 0	OSL	N/A
OSX3	Library-4 OS/X Bank-Switched ROM		SYS	5
OTRP	Oventrop Ventil Module	2 pages	ENG	5/6
P3BC	Aviation Pac for P3B/C	4 pages	GOV	9/21/31
PANA	Paname ROM	2 pages	GEN	5/9
PARI	ProtoPARIO ROM		HWS	14

PCOD	Pcoder 1A Module		HWS	16
PETR	HP Petroleum Fluids Pac 1A	2 pages	ENG	15/16
PHYH	HP-41 Physics Solutions		PHY	15
PKP1	Poul Kaarup’s Alpha and Pointers ROM		UTL	31
PKP2	Poul Kaarup’s Math and Physics ROM, pg 1	2 pages	MAT	14/15
PKP3	Poul Kaarup’s Flags and Stack ROM		UTL	3
PKP4	Poul Kaarup’s Program Utilities ROM		UTL	5
PKP5	Poul Kaarup’s Timer and Utilities ROM		UTL	18
PLOT	HP Plotter Pac 1A	2 pages	HIL	17/18
PMLB	PPC-Melbourne ROM		GEN	12
POLY	Polynomial Functions ROM	2 pages	MAT	6/9
PPCM	PPC Module	2 pages	GEN	10/20
PPCU	PPC User Programs	2 pages	GEN	17/18
PPOK	Poker ROM		GAM	10
PRFS	Profiset ROM	2 pages	HWS	27/31
PRIQ	Pride ROM	2 pages	ENG	21/31
PROG	Program Generator ROM		PRG	18
PRTW	Ports ROM	2 pages	NAV	11
PSOF	PS0F ROM		HWS	16
PSRV	Printer Service ROM	Page 4	OSL	N/A
PWRL	Power CL ROM		UTL	12
PWRX	Power CL Extreme ROM		UTL	12
QUAT	Quaternions ROM	2 pages	MAT	15/16
RAMP	RAMpage ROM		UTL	15
REAL	HP Real Estate Pac 1A	2 pages	FIN	11
RM32	RAMbox-32 ROM		HWS	31
RMPG	RAMpage ROM		UTL	15
ROAM	ROAM-0A ROM		GEN	5
ROMS	ROMSV01 ROM		UTL	9
ROSV	RSU-OS ROM	2 pages	HWS	4/6
RUBK	Rubik’s Cube ROM		GAM	8
SANA	Sandmath-7 ROM (12K version)	4 pages	MAT	2/3/6
SBOX	Sandbox ROM	2 pages	MAT	8/13
SEAK	SeaKing ROM		GOV	21
SECY	HP Securities Pac 1A		FIN	19
SGSG	SGS GAS Module		ENG	21
SHTZ	Spreadsheet ROM		FIN	8
SIHP	SI ROM		MAT	24
SIMM	SIM Module	4 pages	SVY	4/10/30/31

SKWD	SKWID ROM		HIL	8
SM33	Library-4 Sandmath 3x3 ROM	2 pages	MAT	2/3
SM44	Library-4 Sandmath 4x4 ROM	2 pages	MAT	2/3
SMCH	Speed Machine II ROM	2 pages	FIN	21/31
SMPL	Simplex Module		MAT	16
SMTS	Sandmath-7 ROM (8K version)	2 pages	MAT	2/3
SND2	Sandmath II ROM	2 pages	MAT	2/3
SPEC	Spectral Analysis ROM		MAT	8
STAN	HP Standard Applications Pac 1C		GEN	5
STAT	HP Statistics Pac 1B		MAT	2
STEQ	Steam Properties ROM		STEQ	12
STRE	HP Stress Analysis Pac 1A		ENG	8
STRU	HP Structural Analysis Pac 1B	2 pages	ENG	7/19
SUD1	Sudoku ROM		GAM	16
SUPR	SUP-R-ROM	2 pages	SVY	21/31
SURV	HP Survey Pac 1B		SVY	3
TAFB	Tinker AFB ROM	2 pages	GOV	21/31
TDSI	TDS Instrument ROM	2 pages	SVY	5,12
TDSM	TDS Surveying ROM	4 pages	SVY	4,10,31,30
TDSP	TDS Plotter ROM	2 pages	SVY	8,9
TEST	HP-41 Test Statistics Solutions		MAT	13
THER	HP Thermal & Transport Science Pac 1A		ENG	13
TIDW	Tides and Ports ROM		NAV	10
TIME	HP-41 Timer Solutions		GEN	6
TMAX	Turbo-MAX -3A		GAM	6
TMOD	HP-41 Time Module	page 5	OSL	26
TOMS	TOMSROM		SVY	6
TOOL	Toolbox II ROM		UTL	13
TREK	Trekkies ROM		GAM	11
TRIH	83trinh ROM		GEN	9
TTRC	Total Rekall ROM		GEN	20
TVMY	TVM ROM		FIN	22
UCCD	CCD Manual examples		GEN	18
UCLN	User Calendar ROM		GEN	18
UNIT	UnitConv ROM		ENG	10
USPS	USPS Module	2 pages	GOV	21/31
VECT	Vector Analysis ROM		MAT	14
VEGS	Vegas -1C		GAM	6
VERM	Vermpack ROM		SVY	27

VONK	Math Programs Collection		**MAT**	16
WRAM	W&W Rambox-64B ROM		**HWS**	31
WWDB	Wickes, Wlodek, Dearing Books		**GEN**	17
XBFR	Direct Stiffness Method: Beams & Frames	2 pages	**ENG**	30
XFN3	HP-41 X-Functions (page 3)	page 3	**OSL**	25
XFN5	HP-41 X-Functions (page 5, bank 2)	page 5/bnk 2	**OSL**	N/A
XPMM	CL X-Memroy Functions		**OSL**	20
XTAT	XM Statistics		**MAT**	6
XTRS	Direct Stiffness Method: Trusses	2 pages	**ENG**	30
XXXA	4K user image at address 0x0C8000		**GEN**	
XXXB	4K user image at address 0x0D0000		**GEN**	
XXXC	4K user image at address 0x0D8000		**GEN**	
XXXD	8K user image at address 0x0E0000	2 pages	**GEN**	
XXXE	8K user image at address 0x0E2000	2 pages	**GEN**	
XXXF	16K user image (four banks) at address 0x0E4000		**GEN**	
YACH	Bobby Schenk’s Yacht Module	2 pages	**NAV**	21/31
YBFR	Extra Functions Buffer Area (RAM page 805)		**OSL**	N/A
YFNF	41CL Memory Functions		**OSL**	16
YFNP	41CL Extra Functions Plus		**OSL**	15
YFNS	optional 41CL Extra Functions (XROM #31)		**OSL**	31
YFNX	41CL Extreme Functions		**OSL**	15
YFNZ	default 41CL Extra Functions (XROM #15)		**OSL**	15
YLIB	41CL Extreme Functions Library	Page 4	**OSL**	N/A
Z41Z	HP41Z Complex Number ROM	2 pages	**MAT**	1/4
ZENR	ZENROM ROM		**UTL**	5
ZEPM	ZEPROM ROM		**HWS**	9
16CS	16C Simulator ROM		**GEN**	16
2SWP	Misc routines from 412 Swap Disks	2 pages	**GEN**	10
3SWP	Swap Disk Math ROM	2 pages	**MAT**	12/13
41AD	HP Advantage Pac 1B	2 pages	**GEN**	22/24
441Z	Library-4 HP41Z Complex Number ROM	2 pages	**MAT**	1/4
4ADV	Library-4 Advanced Matrix ROM	2 pages	**MAT**	22/24
4ALP	Library-4 Alpha ROM		**UTL**	6
4AOS	Library-4 AMC OSX ROM		**SYS**	5
4DIG	Library-4 41Z Diagnostic ROM		**MAT**	8
4LIB	Library-4 ROM	Page 4	**PRG**	N/A
4MTI	Library-4 Matrix/Polynomial ROM	2 pages	**MAT**	22
4MTR	Library-4 Matrix ROM		**MAT**	7
4PLY	Library-4 Polynomial ROM		**MAT**	6

4RAM	Library-4 RampageX ROM		**UTL**	17
4SM4	Sandmath44 2x2 ROM	2 pages	**MAT**	2/3
4SMT	Library-4 Sandmath ROM	2 pages	**MAT**	2/3
4TBX	Library-4 Toolbox ROM		**UTL**	13
4UTL	Library-4 CL Utilities ROM		**UTL**	12

Identifiers highlighted in blue are obsolete, having been superceeded by newer images. These images are not loaded onto V2 boards by default.

Identifiers highlighted in cyan are not loaded onto V2 boards by default because of space constraints.

Group definitions are shown 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

Memory Management

The original 41C system used dedicated ROM and RAM chips to implement the memory for the calculator, and the memory organization was mostly hidden from the user. The 41CL calculator replaces these custom memory chips with a pair of industry-standard memory devices, and besides the normal 41C view of memory, provides built-in functions that allow the user direct access to the physical memory.

The original ROM memories (including the plug-in ROMs in application pacs) are replaced with a single Flash (non-volatile, but re-programmable) memory, while the RAM chips are replaced with a single low-power RAM device. Given the advance of technology since the design of the original 41C system, these new memory devices provide significantly more storage than the original 41C system could even use. To take advantage of this increased storage capacity, the 41CL design includes a Memory Management Unit (MMU). Plug-in application pacs and peripherals are still supported, but separate application pacs are no longer really necessary.

The MMU takes the memory address, in either the program (ROM) address space or the data (RAM) address space, and translates this address into an address in either the Flash memory or the static RAM. This translation is completely automatic and transparent to the user.

Most users will never need to concern themselves with the operation of the MMU, as the new *41CL Extra Functions* take care of programming the MMU in most cases. However, advanced users may need to understand how the MMU works and is programmed to take full advantage of some features of the 41CL calculator.

The MMU and program addresses

The 41C system uses a 16-bit (64K) program address, which is divided into sixteen pages of 4K each. For many of these pages, there can be up to four “banks”, which are selected under software control during program operation. This natural division of 4K pages is used by the MMU in the 41CL, so that each bank in each page can be mapped by the MMU to a specific absolute address in the Flash memory or RAM on the 41CL circuit board.

To accomplish this mapping, the MMU takes the upper four bits of the program address (which selects the page), plus the two bits which select the bank, and forms a special memory address. The contents of this memory location, if the MMU is enabled, replaces the original upper four bits of the program address and forms a 24-bit address that is used to access the memory devices on the 41CL circuit board (the lower twelve bits are not modified by the MMU). Thus, in theory, any page can be mapped to any 4K page in the physical memory on the 41CL circuit board.

The 41CL does not allow user control of the mapping of some of the pages, to protect the Operating System (OS) of the calculator. So pages 0, 1, and 2 are normally never mapped by the MMU, because this is where the basic OS is stored. In addition, pages 3 and 5 are normally never mapped by the MMU, because this is where the X-Functions and Time Module functions for the calculator are located. But every other page can be mapped using the MMU.

The MMU entries are located in the 4K page of memory starting at address 0x804000, which is located in RAM. The actual address for an MMU entry is formed using this base address, with the page number replacing bits 7-4 and the bank number replacing bits 3-2. Note that the order of the banks follows the encoding used by the original 41C hardware, rather than conventional encoding. Since not all pages support banks only the following MMU entries are valid:

Physical Address	Contents
0x80400C	MMU register for Page 0 (no banking supported)
0x80401C	MMU register for Page 1 (no banking supported)
0x80402C	MMU register for Page 2 (no banking supported)
0x80403C	MMU register for Page 3 (no banking supported)
0x804040	MMU register for Page 4, Bank 1
0x804044	MMU register for Page 4, Bank 3
0x804048	MMU register for Page 4, Bank 2
0x80404C	MMU register for Page 4, Bank 4
0x804050	MMU register for Page 5, Bank 1
0x804054	MMU register for Page 5, Bank 3
0x804058	MMU register for Page 5, Bank 2
0x80405C	MMU register for Page 5, Bank 4
0x804060	MMU register for Page 6, Bank 1
0x804064	MMU register for Page 6, Bank 3
0x804068	MMU register for Page 6, Bank 2
0x80406C	MMU register for Page 6, Bank 4
0x804070	MMU register for Page 7, Bank 1
0x804074	MMU register for Page 7, Bank 3

0x804078	MMU register for Page 7, Bank 2
0x80407C	MMU register for Page 7, Bank 4
0x804080	MMU register for Page 8, Bank 1
0x804084	MMU register for Page 8, Bank 3
0x804088	MMU register for Page 8, Bank 2
0x80408C	MMU register for Page 8, Bank 4
. .	. .
0x8040F0	MMU register for Page F, Bank 1
0x8040F4	MMU register for Page F, Bank 3
0x8040F8	MMU register for Page F, Bank 2
0x8040FC	MMU register for Page F, Bank 4

Pages 8-F correspond to the Ports on the calculator, with pages 8-9 being Port 1, pages A-B being Port 2, and so on. The MMU entries for these pages are automatically handled by *41CL Extra Functions*, so only the MMU entries for Page 4 needs to be manually programmed.

Page 4 is special to the OS, and only a few ROM images can be used in it. If you don't know what you are doing, don't try to use Page 4. Page 6 is normally used by a printer and Page 7 is used by HP-IL, so don't try to use either of these pages unless you don't need access to a printer or HP-IL.

Pages 0-3 and 5 contain the OS for the machine, so the MMU entries for these pages are normally not used.

The contents of an MMU memory location are used as follows:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EN	LCK	MULTI		A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12

Bit 15 is the Enable (EN) bit. If this bit is zero, the MMU entry is ignored and the corresponding page and bank will be fetched from a Port.

Bit 14 is the Lock (LCK) bit, used only with the 41C Extreme Functions. If this bit is set to one the entry cannot be changed, except with either the ***UNLOCK*** function or the ***MMUCLR*** function.

Bits 13 and 12 are the Multi-page Image (MULTI) bits, used only with the *41CL Extreme Functions*. These bits are managed automatically by the ***PLUG*** and ***PPLUG*** functions, with the following meanings:

13	12	MULTI
0	0	Not part of a multi-page image
0	1	First page of a multi-page image
1	1	Middle page of a multi-page image
1	0	Last page of a multi-page image

Bits 11-0 hold the twelve address bits to be substituted for the Page address (bits 15-12) portion of the logical address, to create the physical address.

The MMU and data addresses

Data addresses (“registers” in 41C parlance) are also translated by the MMU, but this translation is not programmable. Instead, registers are mapped to specific locations in the RAM on the 41CL board. This dedicated mapping is shown in the table below. Note that the 41C OS is not capable of addressing registers above 0x3FF, but space is reserved in the 41CL memory for register addresses up to 0xFFF in case there are future enhancements to the OS code.

In the 41C system only the lower four bits of the register address can be specified by an instruction, and all of the upper register address bits are held in a dedicated register called (not surprisingly) the “register address”. In the 41CL this register address is stored in the RAM in a special location, at address 0x804000.

Like the original 41C, the 41CL preserves the data in RAM as long as power is applied. Unlike the original 41C, the 41CL allows RAM to be used to hold program data. All that is required for this type of operation is that the MMU point to a 4K block of RAM rather than a 4K block in the Flash portion of the address space. In the 41CL bit 23 of the physical memory address determines whether the address is in Flash (bit 23 is zero) or in RAM (bit 23 is one).

A seven-byte 41C register is stored in four successive memory locations as shown below:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
addr lsb																
00	Byte 1								Byte 0							
01	Byte 3								Byte 2							
10	Byte 5								Byte 4							
11	unused								Byte 6							

The table below shows the organization of the 41C register memory in the physical memory of the 41CL circuit board. Because the OS manages the data in these locations, users are discouraged from attempting to modify any of these memory locations.

This table is a subset of the information presented in the mem_ref.pdf document. Refer to that document for a full explanation of how information is organized in the physical memory of the 41CL.

Physical Address	Contents	Operating System (OS) use
0x800000 - 0x800003	Register 000	T register
0x800004 - 0x800007	Register 001	Z register
0x800008 - 0x80000B	Register 002	Y register
0x80000C - 0x80000F	Register 003	X register
0x800010 - 0x800013	Register 004	LAST X register
0x800014 - 0x800017	Register 005	ALPHA register 1-7
0x800018 - 0x80001B	Register 006	ALPHA register 8-14
0x80001C - 0x80001F	Register 007	ALPHA register 15-21
0x800020 - 0x800023	Register 008	ALPHA register 22-28
0x800024 - 0x800027	Register 009	Temp ALPHA Scratch
0x800028 - 0x80002B	Register 00A	Unshifted Key Assign, OS status
0x80002C - 0x80002F	Register 00B	Program Return Stack
0x800030 - 0x800033	Register 00C	Program Return Stack, Program Pointer
0x800034 - 0x800037	Register 00D	Address Pointers
0x800038 - 0x80003B	Register 00E	FLAG register
0x80003C - 0x80003F	Register 00F	Shifted Key Assign, Program Line Number
0x800040 - 0x8000FF	Registers 10 - 3F	not visible to OS
0x800100 - 0x800103	Register 040	X memory
0x800104 - 0x800107	Register 041	
. .	. .	
0x8002F8 - 0x8002FB	Register 0BE	
0x8002FC - 0x8002FF	Register 0BF	
0x800300 - 0x800303	Register 0C0	Main memory
0x800304 - 0x800307	Register 0C1	
. .	. .	
0x8007F8 - 0x8007FB	Register 1FE	
0x8007FC - 0x8007FF	Register 1FF	
0x800800 - 0x800803	Register 200	not visible to OS

0x800804 - 0x800807	Register 201	X memory
0x800808 - 0x80080B	Register 202	
. .	. .	
0x800BB8 - 0x800BBB	Register 2EE	
0x800BBC - 0x800BBF	Register 2EF	
0x800BC0 - 0x800C03	Registers 2F0 - 300	not visible to OS
0x800C04 - 0x800C07	Register 301	X memory
0x800C08 - 0x800C0B	Register 302	
. .	. .	
0x800FB8 - 0x800FBB	Register 3EE	
0x800FBC - 0x800FBF	Register 3EF	
0x800FC0 - 0x800FFF	Registers 3F0 - 3FF	not visible to OS
0x801000 - 0x801FFF	Registers 400 - 7FF	not currently utilized by OS
0x802000 - 0x802FFF	Registers 800 - BFF	
0x803000 - 0x803FFF	Registers C00 - FFF	
0x804000		OS Register Address buffer

Programming the MMU

The **PLUG** functions allow the user to insert module images into nearly every open Page on the 41CL. The one exception is Page 4. This was done intentionally, because Page 4 is special as far as the Operating System (OS) is concerned, and can take over the machine in certain circumstances. But there are a number of images present in Flash memory that can only be loaded into Page 4, and this section will show you how to do this for three of them.

Library-4

The Library-4 ROM was written by ‘Angel Martin specifically for use with several other images that he contributed to the 41CL. The Library-4 ROM contains common subroutines that are called from these other ROMs. Having a fixed address for subroutines allows for much quicker access and more efficient code because the subroutine entry addresses can be hard-coded in the calling programs.

The Library-4 ROM image is located at address 0x120000, and has an associated IMDB entry so that it can be used with the **IMDB?** function. Once installed, it is completely invisible as far as the OS is concerned, even when not in use.

The Library-4 ROM is installed by simply writing to the Page 4 MMU entry, using the **YPOKE** command:

ALPHA 804040-8120 ALPHA
XEQ ALPHA YPOKE ALPHA

The FORTH ROM

The regular **PLUG** functions cannot be used to insert the FORTH ROM into the 41CL logical memory because this ROM is hard-coded to use Page 4 and Page 7. Instead, you will need to program the MMU entries for Pages 4 and 7 directly.

It’s a little more complicated than just programming the two MMU entries though. The problem is that between programming the two MMU entries the OS is going to check certain locations in program memory, including the start of Page 4 and places near the end of Pages 5 through F. This means that the OS will get confused with only half of the FORTH ROM visible. The way around this problem is to disable the MMU during programming. The sequence of commands shown below will enable FORTH ROM.

First, the MMU is disabled. This has no effect on the contents of the MMU:

XEQ ALPHA MMUDIS ALPHA

Next, the MMU for pages 4 and 7 are programmed to point at the FORTH ROM image:

ALPHA 804040-809A ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 804070-809B ALPHA
XEQ ALPHA YPOKE ALPHA

Finally, the MMU is re-enabled:

XEQ ALPHA MMUEN ALPHA

Disabling the FORTH ROM is only possible by turning off the calculator and momentarily removing the batteries, because once the FORTH module is active the 41CL Extra Functions are no longer available.

The HP Service ROM

The HP Service ROM was used by HP to test returned calculators, and is hard-coded to use Page 4. This ROM is intended to take over the calculator, so installation is as simple as programming the MMU for Page 4. Refer to the HP 41C Service Manual for instructions on using the test facilities in this ROM.

ALPHA 804040-8004 ALPHA
XEQ ALPHA YPOKE ALPHA

Disabling the HP service ROM is only possible by turning off the calculator and momentarily removing the batteries, which disables the MMU.

Image Database

An entry in the Image Database consists of four words, and the format of the database is set up to make adding new entries very easy. An unprogrammed Image Database entry contains 0xFFFF in all four words. This is the default contents of Flash memory, where the database is stored, which means that new entries can simply be added to the Image Database without first needing to erase the Flash sector where the database is stored.

The **PLUG** functions will return a ***NO ENTRY*** error message if the user attempts to use a module identifier corresponding to an unprogrammed database entry. An Image Database entry can be "erased" by writing 0x0000 to the first two words. In this case the **PLUG** functions will return a ***NULL ENTRY*** error message for the corresponding module identifier.

To preserve backwards-compatibility with previous versions of the *41CL Extra Functions*, the Image Database entries are addressed as a function of the first and fourth characters of the module identifier only.

In order to limit the size of the database to 4K words only the characters ***A*** - ***Z***, ***1*** - ***5*** and ***9*** (32 possibilities) are allowed for the first and fourth character of a module identifier. In addition, when the first character of the module identifier is ***9*** and only the characters ***A*** - ***Z*** are allowed for the fourth character of the module identifier, to provide some space for housekeeping in the database. Any character is allowed for the second and third characters of the module identifier.

The contents of each database entry are shown in the table below.

Image Data base entry word	Address LSBs	digit 3 meaning	digit 2 meaning	digit 1 meaning	digit 0 meaning
1	00	image group		image type	address<5>
2	01	page restriction	type modifier	address<4>	address<3>
3	10	always 0	always 0	character 3 of module identifier	
4	11	always 0	always 2	character 2 of module identifier	

Digits 3 and 2 of the first word in a database entry are used only by the *41CL Extreme Functions*, to search the database by group. The table below shows the groups available and their encoding in these two digits. Note that these two digits are always written as 00 when using the **IMDBINS** function, so if you want to add a custom identifier to a group, you will need to write directly to the correct memory location (in the RAM copy of the Image Database) with one of these values included.

word 1 digits 3:2	Group	meaning
00	***NUL***	none (or Null Entry)
10	***GEN***	General-Purpose
20	***MAT***	Mathematics
30	***ENG***	Engineering
40	***FIN***	Financial
50	***GAM***	Games
60	***UTL***	Utilities
70	***SYS***	System Extensions
80	***HIL***	HP-IL
90	***HWS***	Hardware-Specific
A0	***OSL***	OS/CL
C0	***AST***	Astronomy
C1	***AVI***	Aviation
C2	***CHM***	Chemistry
C3	***MED***	Medicine
C4	***GOV***	Covernment/Military
C5	***NAV***	Navigation
C6	***PHY***	Physics
C7	***PRG***	Programming
C8	***SVY***	Surveying
Fx	***UNP***	Unprogrammed database entry

Digit 1 in the first word of a database entry specifies the type of image, according to the table below. Only these values are currently valid as far as the **PLUG** functions are concerned.

word 1 digit 1	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)

A type digit of 4 is a slightly special case. The **PLUG** functions treat this image type as 32K words, consisting of four banks to be loaded into two adjacent pages. However, the images that use this type identifier really only use the first two or three banks in the second page. This leaves one or more 4K-word sections of memory available to store other images, and the 41CL takes advantage of this space. So the database search functions treat a type digit of 4 as 24K words in length, and return search results accordingly.

Digit 0 of the first word and digits 1 and 0 of the second word of a database entry hold the starting memory address for the image referenced by the module identifier. This is a physical address that can be in either Flash memory or RAM. Note that if the starting address is 000, along with a type digit of 0, the entry is considered a null entry.

Digit 3 of the second word of a database entry is used only by the *41CL Extreme Functions* if there are restrictions on where the image can be placed, according to the table below:

word2 digit 3	Page Restriction
0	No Restriction
1	Not Pluggable
2	Page 0 only
3-F	Page 3-F only

Digit 2 of the second word of a database entry is used only by the *41CL Extreme Functions* to modify the type field, according to the table below:

word2 digit 2	Type Modification
0	No Modification
1	Modify type 3 to 12K (3 pages)

Digits 1 and 0 of the third and fourth words of a database entry hold the middle two characters of the module identifier. This allows an address-based search of the database to return the full module identifier. It would also allow the **PLUG** functions to check for the full module identifier, but this feature was not implemented to preserve backwards-compatibility. These two words can use either the “display” encoding, where ***A*** - ***Z*** are encoded as 0x41-0x5A, or the “assembly language” encoding, where ***A*** - ***Z*** are encoded as 0x01-0x1A. All of the original entries in the Image Database use the “assembly language” encoding, while any user-added entries will always use the “display” encoding.

As mentioned previously, entries in the Image Database are addressed using the first and fourth characters of a module identifier. Each of these characters must be translated to a 5-

bit field to create an address to index the database. The table below shows the translation algorithm.

character	41 code	address field
A	41	00000
B	42	00001
C	43	00010
D	44	00011
E	45	00100
F	46	00101
G	47	00110
H	48	00111
I	49	01000
J	4A	01001
K	4B	01010
L	4C	01011
M	4D	01100
N	4E	01101
O	4F	01110
P	50	01111
Q	51	10000
R	52	10001
S	53	10010
T	54	10011
U	55	10100
V	56	10101
W	57	10110
X	58	10111
Y	59	11000
Z	5A	11001
1	31	11010
2	32	11011
3	33	11100
4	34	11101
5	35	11110
9	39	11111

The address for an Image Database entry is formed as shown below:

Image Database address nibble 2				Image Database address nibble 1				Image Database address nibble 0			
3	2	1	0	3	2	1	0	3	2	1	0
First character module identifier address field					Fourth character module identifier address field					word identifier	

All versions of the **PLUG** functions (in the *41CL Extra Functions*, the *41CL Extra Functions Plus*, and the *41CL Extreme Functions*) parse only the first and last characters of an image identifier. The tables below present all of the image identifiers organized in columns based on the first character, and rows based on the last character.

	A	B	C	D	E	F	G	H	I
A	AVIA		CCDA	DAVA		FINA			
B	ASMB	B52B				FLDB			IMDB
C		BBSC	CIRC	DA4C		FACC		HVAC	
D		BLND	CLND	DMND	EEFD				ICOD
E	AFDE					FFEE	GAME	HOME	
F	AFDF			DIFF					ILBF
G	ALGG		CENG		EENG				INTG
H	ALPH				EPRH	FRTH			
I		BASI	CIVI	DEVI	EXTI		GRVI		
J									
K		BLJK	CVPK	DYRK					
L		BESL	CNTL	DIIL	ESML			HCPL	ISOL
M			CHEM	DASM	EPRM		GEOM		
N	AFIN		CLIN		EPTN	FDYN	GTWN		ISEN
O	AUTO		CRTO	DEMO	EXIO				INDO
P	A41P		CCDP		EILP			HCMP	ITCP
Q	AV1Q		COOQ	DACQ					
R	AECR	BLDR	CCDR			FAIR		HEPR	IERR
S	ANTS	BSMS	CHES		E41S	FUNS	GMAS	HDIS	
T	ASTT		CLUT	DIGT		FCST	GMAT	HMAT	
U			CIVU						
V	AADV		CURV				GSLV	HSRV	
W		BCMW					GRAW		
X	AOSX	BJMX	CCDX					HEPX	IBOX
Y	ALGY					FSSY	GMTY		
Z	ANGZ								
1	ADV1		CMT1	DST1					IDC1
2	ADV2	BUD2	CMT2	DEV2		FCS2		HEP2	IDC2
3	AEC3	BUD3	CMT3		ETS3				
4	ASM4				ETS4				
5					ETS5				
9					ETS9				

	J	K	L	M	N	O	P	Q	R
A				MTRA			PANA		
B				MELB			PMLB		
C	JMBC		LPLC	MUEC	NPAC		P3BC		
D			LAND				PCOD		
E				MILE					
F							PSOF		
G			LENG	MENG			PROG		RMPG
H				MATH			PHYH		
I				MLTI	NAVI		PARI		
J									
K				MCCK			PPOK		RUBK
L			LNDL	MLBL	NONL		PWRL		REAL
M		KRGM		MLRM	NVCM		PPCM		ROAM
N				MCHN					
O				MONO		OPTO			
P				MCMP	NCHP	OTRP			RAMP
Q							PRIQ		
R	JARR			MBFR	NFCR		PETR		
S			LBLS	MASS			PRFS		ROMS
T	JMAT		LAIT	MTST	NEXT		PLOT	QUAT	
U							PPCU		
V				MADV			PSRV		ROSV
W						OILW	PRTW		
X	JMTX			MTRX					
Y					NTHY		POLY		
Z						OBCZ			
1				MDP1	NEA1	OS41	PKP1		
2				MDP2			PKP2		RM32
3				MWK3	NEA3	OSX3	PKP3		
4				MWK4			PKP4		
5		K135			NEA5		PKP5		
9			L119						

	S	T	U	V	W	X	Y	Z	-
A	SANA					XXXA			
B		TAFB			WWDB	XXXB	YLIB		
C	SPEC	TTRC				XXXC			
D	SKWD	TMOD	UCCD			XXXD			
E	STRE	TIME				XXXE			
F						XXXF	YFNF		
G	SGSG								
H	SMCH	TRIH					YACH		
I		TDSI							
J									
K	SEAK	TREK		VONK					-16K
L	SMPL	TOOL							-DBL
M	SIMM	TDSM		VERM	WRAM	XPMM		ZEPM	-RAM
N	STAN		UCLN						
O									
P	SIHP	TDSP					YFNP		
Q	STEQ								
R	SUPR	THER				XBFR	YBFR	ZENR	
S	SMTS	TOMS	USPS	VEGS		XTRS	YFNS		
T	STAT	TEST	UNIT	VECT		XTAT			
U	STRU								
V	SURV								
W		TIDW							
X	SBOX	TMAX					YFNX		-MAX
Y	SECY	TVMY							
Z	SHTZ						YFNZ	Z41Z	
1	SUD1								
2	SND2								
3	SM33					XFN3			
4	SM44								
5						XFN5			
9									

	1	2	3	4	5	9
A						
B				4LIB		
C						
D						
E						
F						
G				4DIG		
H						
I				4MTI		
J						
K						
L				4UTL		
M				4RAM		
N						
O						
P		2SWP	3SWP	4ALP		
Q						
R				4MTR		
S	16CS			4AOS		
T				4SMT		
U						
V				4ADV		
W						
X				4TBX		
Y				4PLY		
Z				441Z		
1						9MM1
2						9DD2
3						9YY3
4				4SM4		
5						
9						

Identifiers highlighted in blue are obsolete, having been superceeded by newer images. These images are not loaded onto V2 boards by default, and may be removed from the standard Flash memory map inthe future.

Identifiers highlighted in cyan are not loaded onto V2 boards by default because of space constraints.

Identifiers highlighted in red are deprecated.

Idenifiers 9MM1, 9DD2 and 9YY3 are not really identifiers. Rather, the MM/DD/YY are replaced with the issue date of the Image Database.

Patching Code

The *41CL Extra Functions* make it simple to patch software pre-loaded into the 41CL. Most of the software pre-loaded into the Flash memory can be copied to the RAM memory, patched, and then the MMU can be used to reference this patched code. Only pages that cannot be relocated by the MMU cannot be patched. This is pages 0-3 and 5 (which hold the Operating System, the Extended Functions and the Time Functions.) As mentioned previously, these pages are protected from modification to prevent users from inadvertently turning the 41CL into a brick.

To illustrate what is required to patch code, go through the steps below to modify the ROM ID of the *41CL Extra Functions* to avoid a conflict with other modules.

To patch code the ROM image must first be copied to an available page in RAM using the **YMCPY** function. The **YMCPY** function automatically executes in 50x Turbo mode and requires about 8 seconds to complete.

ALPHA 007>80C ALPHA
XEQ ALPHA YMCPY ALPHA

In this example just one location needs to be modified for proper operation. Note that whenever patches are specified only the 4K relative address will be given. Use the upper nibbles of the RAM address chosen to hold the patched code for the remainder of the address.

0x000 should be 0x0010 to change the ROM ID to 16 (which is 10 in hexadecimal)

Use the **YPOKE** function to write directly to the desired location in RAM memory. Since we are using the page starting at address 0x80C000 to hold the patched ROM image the following keystrokes are required to apply this patch:

ALPHA 80C000-0010 ALPHA
XEQ ALPHA YPOKE ALPHA

Then we can use the **PLUG1L** function to insert the patched image into the lower half of Port 1 (or wherever the 41CL Extra Functions are, or will be, located):

ALPHA 80C-RAM ALPHA
XEQ ALPHA PLUG1L ALPHA

It’s that simple! The MMU in the 41CL, along with the ability to peek and poke memory, makes this machine a hacker’s delight.

Using HEPAX

The HEPAX module was one of the most complex third-party 41C modules ever created. It had hardware and software that automatically relocated the module to the lowest unused page of memory, and provided RAM that resided in program memory for file storage. Not all of the features of the HEPAX module are supported by the 41CL calculator, and this section will discuss the details of how to use the HEPAX image included in the 41CL Flash memory.

The HEPAX module implemented write-protection for pages of HEPAX RAM, using an opcode that was ignored by the processor in the 41C. This opcode is also ignored by the NEWT processor in the 41CL, but the write-protect feature is not supported by the 41CL hardware. Keep this in mind if you attempt to write-protect pages of HEPAX memory.

The *41CL Extra Functions* make it easy to create a backup copy of RAM pages, using the **YMPCY** function to copy an entire page of memory to another location in physical memory. If you are doing work that might inadvertently corrupt HEPAX RAM, which is what the write-protect feature was for, try creating a copy of the HEPAX RAM page first. Since the HEPAX code does not “know” about physical memory, the copy will effectively be write-protected.

The main issue for HEPAX users is that the 41CL hardware does not support the automatic relocation of the HEPAX image. The MMU makes this function unnecessary, so the HEPAX image in the 41CL has been modified to eliminate the automatic relocation feature. Unfortunately a by-product of this modification is that the HEPAX RAM will not be automatically initialized at start-up. Instead, you will need to initialize any HEPAX RAM when the HEPAX image is first plugged into a port.

The 41CL provides a template for initializing HEPAX RAM pages, which simplifies the process considerably, but you will still need to initialize one or two locations in each RAM page if you are using multiple pages of HEPAX RAM. If you want to use just one page of HEPAX RAM, no extra initialization is required.

The 41C code listing below shows the template for initializing HEPAX RAM. This template, which is stored at address 0x0B9000 in the 41CL Flash, should be copied to each 4K block of RAM that is going to be used for HEPAX RAM. The “fixed value” locations are checked by the HEPAX software, and values not matching those shown in the listing template will cause an error when the HEPAX code attempts to use the RAM.

Two locations in each HEPAX RAM page contain address pointers. These address pointers hold four-bit page numbers in the least-significant digit. The pointer at address 0xFE7 points to the previous page of HEPAX RAM, and a value of 0x000 here marks a page of HEPAX RAM as the first in the chain. The pointer at address 0xFE8 points to the next page of HEPAX RAM, and a value of 0x000 here marks a page of HEPAX RAM as the last in the chain.

```
;*************************************************************************
      .TITLE    "HEPAX RAM"

      .HP

      XROM 13

      .FILLTO   0FE6

      #000      ; FE7 Previous page identifier
      #000      ; FE8 Next page identifier
      #091      ; FE9 fixed value
      #000      ; FEA
      #000      ; FEB
      #000      ; FEC
      #090      ; FED fixed value
      #000      ; FEE
      #091      ; FEF fixed value
      #000      ; FF0
      #0E5      ; FF1 fixed value
      #00F      ; FF2 fixed value
      #200      ; FF3 fixed value

      .FILLTO   0FFE
```

As an example, the sequence of commands listed below uses four pages of 41CL RAM (at addresses 0x808000, 0x809000, 0x80A000 and 0x80B000) as HEPAX RAM assigned to pages C through F (Ports 3 and 4).

First, the RAM is initialized by copying the HEPAX RAM template to these four pages of RAM memory:

ALPHA 0B9>808 ALPHA
XEQ ALPHA YMCPY ALPHA

ALPHA 0B9>809 ALPHA

XEQ ALPHA YMCPY ALPHA

ALPHA 0B9>80A ALPHA
XEQ ALPHA YMCPY ALPHA

ALPHA 0B9>80B ALPHA
XEQ ALPHA YMCPY ALPHA

Next, the HEPAX RAM pointers in these pages must be initialized. The template loads 0x0000 into all of the pointers to start with, so only the non-zero pointers are written:

ALPHA 808FE8-000D ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 809FE7-000C ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 809FE8-000E ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 80AFE7-000D ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 80AFE8-000F ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 80BFE7-000E ALPHA
XEQ ALPHA YPOKE ALPHA

Finally the RAM pages are plugged into the Ports:

ALPHA 808-RAM ALPHA
XEQ ALPHA PLUG3L ALPHA

ALPHA 809-RAM ALPHA
XEQ ALPHA PLUG3U ALPHA

ALPHA 80A-RAM ALPHA
XEQ ALPHA PLUG4L ALPHA

ALPHA 80B-RAM ALPHA
XEQ ALPHA PLUG4U ALPHA

At this point the HEPAX RAM is initialized to a point where the HEPAX code can recognize and use it. Next **PLUG** the HEPAX image into a Port. To verify that you've done everything correctly, try executing a HEPDIR command. The display should return ***H:DIR EMPTY***, and clearing this from the display should show ***2610***, which is the size of four pages of HEPAX RAM.

Patching HEPAX

The HEPAX **DISASM** function scans the keyboard during the disassembly process, but it appears that to save space this scanning function was not implemented properly. (The code does not look at the keyboard valid flag.)

In addition, early versions of the 41CL keyboard scanner did not output the same scan code as the original 41C when no key is being pressed. (The idle state code was not specified in the HP documentation.)

As a result of these two issues the HEPAX **DISASM** code thinks that the **ON** key has been pressed immediately after the last address digit has been entered, turning the calculator off.

The way around this issue is to remove the test for a press of the **ON** key during the HEPAX **DISASM** function. This requires copying one page of the HEPAX code to RAM so that one location can be patched, and then pointing the MMU at the patched code.

The example below assumes that the HEPAX module has been loaded into the lower half of Port 2, which is page A, and that the uppermost page of RAM (starting address 0x83F000) will be used for the patched HEPAX page.

First, the Bank 4 HEPAX image is copied to RAM:

ALPHA 030>83F ALPHA
XEQ ALPHA YMCPY ALPHA

Next, the instruction that tests for a press of the **ON** key is replaced with a NOP instruction:

ALPHA 83F08D-0000 ALPHA
XEQ ALPHA YPOKE ALPHA

Finally, this RAM page is substituted for bank 4 of the HEPAX image in Flash by directly programming the MMU register. The MMU register must be programmed directly because we are only substituting one bank of the HEPAX code.

ALPHA 8040AC-883F ALPHA
XEQ ALPHA YPOKE ALPHA

Enabling HEPAX Disassembly

The HEPAX **DISASM** function does not allow the disassembly of the HEPAX code itself. If you want to remove this restriction, four locations in Bank 1 of the HEPAX code need to be modified. The code is in Bank 1 of the HEPAX code, and we will use the RAM at address 0x83E000 to hold the patched code.

First, the Bank 1 HEPAX image is copied to RAM:

ALPHA 02D>83E ALPHA
XEQ ALPHA YMCPY ALPHA

Next, the instructions that branch to an error routine are replaced with NOP instructions:

ALPHA 83E131-0000 ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 83E132-0000 ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 83E133-0000 ALPHA
XEQ ALPHA YPOKE ALPHA

ALPHA 83E134-0000 ALPHA
XEQ ALPHA YPOKE ALPHA

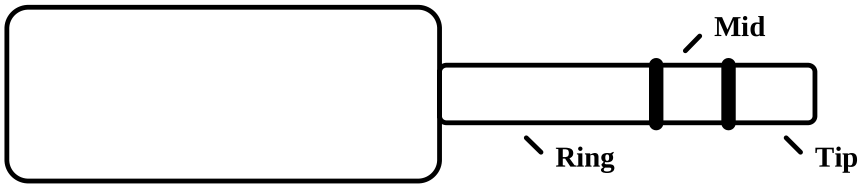
Finally, this RAM page is substituted for bank 1 of the HEPAX image in Flash by directly programming the MMU register:

ALPHA 8040A0-883E ALPHA
XEQ ALPHA YPOKE ALPHA

Serial Connector

The serial connector jack signals are assigned as follows:

- Tip: Transmit Data from the point of view of the 41CL calculator. This should connect to pin 2 of a female DB9 connector for use with a PC.
- Mid: Ground. This should connect to pin 5 of a female DB9 connector for use with a PC.
- Ring: Receive Data from the point of view of the 41CL calculator. This should connect to pin 3 of a female DB9 connector for use with a PC.



The type of cable required to connect the 41CL calculator to a PC is also used for older digital cameras and cell phones, so it can still be found. However, be aware that two different signal arrangements were used for these types of cables, depending on the manufacturer. Part number BC20213-6 from www.cableclub.com is compatible with the 41CL calculator.

The RS-232 driver on the 41CL board normally only powers up when a vaild level is detected on the Receive Data input. This can make 41CL-to-41CL serial transfers complicated. Starting with Version 4, the 41CL board supports a way for software to force the RS-232 driver on unconditionally. So if one of the 41CL calculators involved in a 41CL-to-41CL transfer supports this feature the transfer is simple. All that is required is a null-modem adapter connecting the two serial cables. Part number 1202 from www.monoprice.com is a suitable adapter.

If you want to try to construct an internal serial connector yourself the part numbers are listed below. Note that the cable comes with 10 conductors, so you will have to trim it down to three conductors. Refer to the 41CL schematic for the signal connections on the circuit board connector.

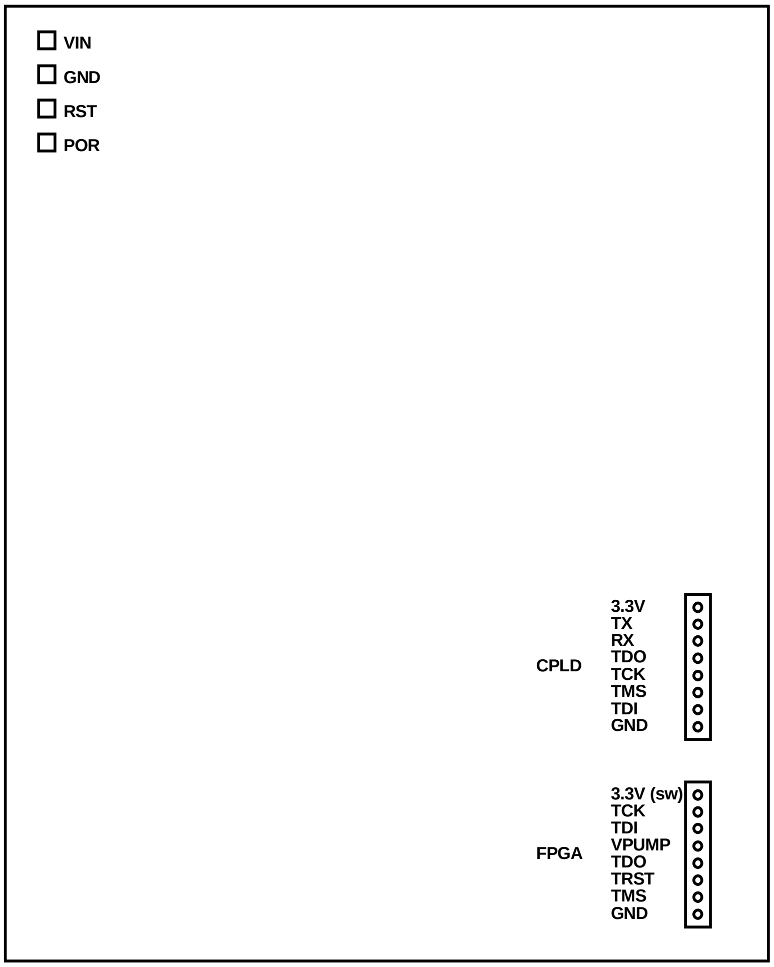
Plug for circuit board connector: 455-2189-ND from Digi-Key

Multi-conductor cable: MB20G-10-ND from Digi-Key

2.5mm Stereo Jack: 161-7000-EX from Mouser

Updating 41CL Hardware

The 41CL board uses programmable logic. This means that with the right equipment the hardware can be updated to correct errors. The same facilities that allow hardware programming can also be used to update the Flash memory on the board. This section will describe the connections necessary to perform hardware programming. The figure below shows the top side of the 41CL board.



All programming operations require that the board be powered. Using the normal HP41 connector on the bottom of the board is not an appropriate way to power the board for programming. Instead, there are three solder pads in the corner opposite the programming connectors that need to have wires soldered to them.

Solder short (20-30cm should be sufficient) wires to the solder pads labelled VIN, GND and POR. The RST solder pad is not required for programming.

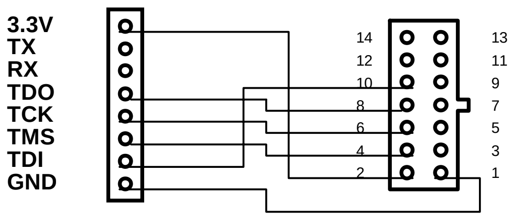
Battery power is not appropriate from programming. Instead use a benchtop power supply. The 41CL board requires 5V (4.0v minimum / 6.0V maximum) for programming.

- Make sure the power supply is OFF before connecting it to the 41CL board.
- Connect the GND wire to the ground of a benchtop power supply.
- Connect the VIN wire to the positive output from the power supply.
- Connect the POR wire to the ground of the power supply to program the FPGA or the Flash. Leave this wire unconnected to program the CPLD. Grounding the POR signal turns on the switched power supplies on the board.

Do not turn on the power supply until the programming cable is connected. Be aware that some variable power supplies overshoot quite a bit on start-up. If you are using a variable supply it is probably better to manually ramp up the supply voltage from zero to the final value rather than turning on the supply with the full supply voltage selected.

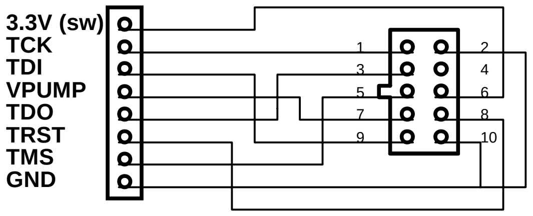
CPLD Programming

The CPLD is programmed using the connector labelled CP on the board. This is the same connector used for the serial port signals. The CPLD is programmed using a Xilinx programming cable. The figure below shows the wiring required between the CPLD connector on the board and the standard Xilinx programming cable.



FPGA Programming

The FPGA is programmed using the connector labelled FP on the board, using an Actel programming cable. The figure below shows the wiring required between the FPGA connector on the board and the standard Actel programming cable.



Flash Programming

The Flash memory is also programmed using the connector labelled FP on the board. The exact connection will depend on the JTAG programming hardware that you use. The important thing to remember is that the TRST signal must be grounded to enable the JTAG controller, and the VPUMP signal must be left floating.

Revision History

Date	Changes	Pages
01/14/2011	Initial release	
01/15/2011	Misc. typos	
01/19/2011	Added chapter on FORTH41, added rampage image	
01/23/2011	more typos	
01/25/2011	more typos	
02/10/2011	more typos, added pictures, serial connector section	
02/22/2011	more typos	
04/04/2011	Added paragraph on backing up 41C register memory Added “Patching Code” section	23 59 on
04/22/2011	Removed “current bank” as logical address option. Added clarification to serial functions. Miscellaneous clarifications to text. Added some new module images and mnemonics. Modified the “Patching Code” section for a different example.	
04/29/2011	Updates to Flash functions, color changes	
04/30/2011	Fixed some error messages (PLUGxx and YFERASE) Function Summary, Error Messages sections added	various 93 on
05/01/2011	Deleted ASTU module mnemonic. Included in ASTT.	18
05/03/2011	Spectral Analysis ROM added	
05/17/2011	Added Algebra, Sandmath II and Modified Advantage modules	
06/06/2011	Corrected XROM number for BCMW ROM. Added cautionary note about serial port connector.	18 59
06/11/2011	Added part list for serial connector	59-60
06/14/2011	Typo in register byte layout	38
06/16/2011	Added info on current drain Added info on Operating System usage of register address space	31 38-39
06/24/2011	Updated recover procedure for lost Y-functions.	12
07/16/2011	Updated Chapters 3, 4 and 5 for YFNS-1C	13-50
08/25/2011	Complete revision	
09/11/2011	typos	
09/16/2011	New modules, flash memory map	
01/10/2012	Added new module images on second batch of boards	43-55
02/02/2012	Clarifications in “Introduction” and “Getting Started” chapters	5-14
02/19/2012	Added new images for Version 3 hardware	18-58
08/13/2012	Updated for YFNS-4A	
09/03/2012	Added Library-4 Sandmath information	

09/28/2012	Added notes about connecting to the serial port.	35
10/05/2012	Revised YFNS/Z to -4B, YFNP to -1B	
12/13/2012	Added new module images and mnemonics	
12/14/2012	Added one more module image	
12/15/2012	Added more Library-4 images	
12/25/2012	Added YCRC values for all images	Ch. 10
12/31/2012	mistake in number of available RAM pages	58
01/01/2013	added note to Printer Service ROM: page 4 takeover ROM	54
01/07/2012	a number of typos (thank you Dan Grelinger for your review)	
01/08/2013	Clarified the differences between YFNZ, YFNS and YFNP	Ch. 2
01/28/2013	PLUGP warning Module Table Memory Buffer Functions explanation Special MMU Functions explanation Added new images, new image versions	19 20-26 31 40 Ch.5, Ch. 10
02/07/2013	Note about MMUCLR not affecting MMU registers for Pages 0-3	16, 41
02/16/2013	Updated YCRC value for new images	Ch. 10
02/22/2013	Added two new images	Ch. 10
02/26/2013	Typos and clarifications throughout	
03/05/2013	More new images...	
04/24/2013	More new images... also, had to move ROSU image because it's actually an 8K image	
05/10/2013	Complete reformat	
05/12/2013	forgot Number Theory ROM in tables	
05/14/2013	serial connector clarification	
05/22/2103	added missing YCRC values	109, 110
05/25/2013	new images	
05/27/2013	new images	
06/10/2013	typos	120
06/19/2013	updated YCRC values for latest images	Ch. 16
07/03/2013	new images, updated YCRC values for new & latest images	Chs. 7, 13, memref
08/19/2013	updated images	Chs. 7, 13, memref
08/27/2013	new/revised images	Chs. 7, 13, memref
09/09/2013	updated YCRC values for new/updated images	memref
11/29/2013	updated YCRC values for new/updated images	memref
12/04/2013	updated YCRC values for updated images	memref
12/05/2013	Expanded descriptions for 41CL Extreme Functions	Ch. 12, 17
12/11/2013	Expanded descriptions for 41CL Extreme Functions	Ch. 6, 7
01/04/2014	updated YCRC values for updated images	memref
02/24/2014	corrected bank numbering in memory reference	memref
03/14/2014	updated YCRC values for updated images, aesthetics	memref
04/01/2014	new images	memref
04/08/2014	new versions: PWRL, PWRX, YFNX	memref
04/28/2014	new versions: YFNX, 4RAM	memref
05/04/2014	added NutIP ROM image	memref, etc.

05/23/2014	new images	memref, etc.
05/28/2014	new versions: TVMY, SM33	memref
06/03/2014	new versions: SM33, YFNX, YLIB	memref
07/10/2014	new images: BASI, FCST, FCS2, COOQ	memref, etc.
07/18/2014	new image: FSSY	memref, etc.
07/27/2014	new images: FFEE, ETS9	memref, etc.
07/31/2014	new images: CIVI, CIVU, VONK	memref, etc.
08/04/2014	new image: NONL	memref, etc.
08/06/2014	new versions: ETS9, NONL	memref
08/12/2014	updated memory reference	memref
08/16/2014	typo (thank you, Gene)	28
08/17/2014	another typo (does it ever end?) new versions: CIVU, ETS5	75 memref
08/23/2014	new versions: PWRX new image: XPMM	75 memref, etc.
08/26/2014	new versions: 4LIB, PWRX	memref
10/06/2014	new versions: PWRX, YFNX	memref
10/16/2014	new version: XPMM	memref
10/17/2014	new version: XPMM	memref
11/22/2014	updated 41CL functions	
11/24/2014	reformat	
12/04/2014	new versions: 4LIB, PWRX, CLUT; updated memref	
12/17/2014	typos, removed memref it’s now a stand-alone document only	
01/18/2015	new images: SM44, ANGZ, TMAX, BLJK, BJMX, VEGS	
02/03/2015	mark obsolete/superceeded images in identifier table	55-62
02/11/2015	new image: PPCU	59, 89
02/17/2015	new identifier: MBFR; clarification for IMDB operation	59, 89, 42
02/22/2015	user feedback (thank you, Bob)	various
02/23/2015	new images: WWDB, JARR, GRAW, MCCK, KRGM, UCCD	various
02/24/2015	new images: PKP1, PKP2, PKP3, PKP5, PKP7	various
02/26/2015	updated identifer information for V2 boards	various
03/01/2015	UCCD XROM number	61
03/28/2015	new image: GTWN, minor changes to V2 image selection	various
04/29/2015	new images: TTRC, 16CS	various
06/01/2015	new images: TDSI, TDSM, TDSP, HCPL	various
09/08/2015	new images: STEQ, 2SWP	various
09/10/2015	updated acknowledgement section	inside cover
09/12/2015	new images: PPOK, TIDW	various
09/15/2015	cleaned up Image Identifer section	
09/05/2015	new image: PRTW	various
10/05/2015	new image: CLND	various
10/26/2015	new images: GRVI, RUBK, TAFB, JBMC, UCLN, EEFD	various
11/16/2015	new images: 3SWP, XBFR, XTRS, CRTO	various
12/07/2015	new/updated images: XTAT, 3SWP, EPTN	various