HP41CL – The Easy Way

Introduction

The HP41CL is a very powerful tool and might be intimidating, as I found out myself. Loads of excellent manuals and tools are available that describe in detail and deep background how to use the power of the 41CL. When starting to use my 41CL I found myself switching between documents and asking what the difference is between (for example) XFNX and XFNZ. So I decided to write down my experiences in a somewhat organized way, with an absolute minimum use of YPEEK and YPOKE commands. I will start with some basic things as setting up the 41CL for the first time, and gradually do more complicated things. How things are done may be very personal, as the HP41CL allows many different ways to achieve the same.

Meindert Kuipers *Email: meindert (at) kuiprs (dot) nl*

Spring 2022

1. Conventions

`0x040`	The 0x sequence is used to indicate hexadecimal values
HP41	HP41 indicates all versions of the HP41 calculator, including HP41C, CV, CX, CL, DM41X
41CL	Is used when specifically referring to the HP41CL system
TEXT	in **blue** is actually typed in in the examples, when quotes are used this means that the text is typed in ALPHA mode. When used in the text it refers to an actual HP41 command
[ALPHA]	when in a bracket this is an HP41 key
0.0000	is the result of an operation, what is in the display in **green**

2. Copyrights and Disclaimer

Photo credits: all photographs by Meindert Kuipers.

3. Version History

Version 1.0	May 2022	First published version

4. Table of Contents

5. Prerequisites

In order to use this document, some previous knowledge is expected:

- Hands on experience with the HP41 ecosystem
- Understanding FOCAL (HP41 User Code), key assignments
- Have a basic understanding of the HP41 memory structure
- Most HP41CL configurations will be based on the HP41CX, including TIME module, Extended Functions and full Extended Memory. So we will assume that that is also your configuration
- I will use examples based on my own HP41CL, which has a V5 CPU board and the maximum amount of memory and a TIME module physically plugged in Port 2

I also assume that you have a working and fully updated 41CL in your hands. Please understand that some of the proposed operations may erase (part of) your existing configuration, so do not be afraid of an occasional **MEMORY LOST** message. It also helps to have a computer available with a serial connection to your HP41CL to do upgrades or transfer data.

FIGURE 1 HAVE YOUR HP41CL READY TO RUN

6. Power of the HP41CL

The power of the 41CL lies in the ability to plug virtual modules in any (virtual) port under user software or keyboard control. But with power comes responsibility, and care must be taken to do the right thing. The open system allows the user to do almost anything, from configuring a legacy configuration, for example with the PPC ROM as I will show, to a microcode powerhouse that the original designers of the HP41 could never have dreamed of. And of course the speed of the 41CL is up to 50 times that of the original HP41, and you get an enormous amount of background memory with a library of nearly all available modules.

7. The view of memory from the HP41 user

Let's start with an overview of the HP41 memory map, this is the foundation of understanding the HP41CL operations. From the perspective of the HP41 the address space is 64K long, split into 16 pages of 4K each. Do not worry about the word size (yet).

Port	Page	Address	Bank 1	Bank 2	Bank 3	Bank 4	HP41 use
4	F	0xF000 – 0xFFFF	[Port 4, upper Page]				
	E	0xE000 – 0xEFFF	[Port 4, lower Page]				cardreader
3	D	0xD000 – 0xDFFF	[Port 3, upper Page]				
	C	0xC000 – 0xCFFF	[Port 3, lower Page]				wand
2	B	0xB000 – 0xBFFF	[Port 2, upper Page]				
	A	0xA000 – 0xAFFF	[Port 2, lower Page]				
1	9	0x9000 – 0x9FFF	[Port 1, upper Page]				
	8	0x8000 – 0x8FFF	[Port 1, lower Page]				
	7	0x7000 – 0x7FFF	[HP-IL Module]				
	6	0x6000 – 0x6FFF	[PRINTER/IL Printer]	[IR PRINTER]			
	5	0x5000 – 0x5FFF	[TIME MODULE]	[CX FNS - bank 2]			
	4	0x4000 – 0x4FFF	[TAKE OVER ROM] [disabled IL Printer]				
	3	0x3000 – 0x3FFF	[CX FNS – bank 1]				
	2	0x2000 – 0x2FFF	HP41CX OS – ROM2				
	1	0x1000 – 0x1FFF	HP41CX OS – ROM1				
	0	0x0000 – 0x0FFF	HP41CX OS – ROM0				

TABLE 1 - HP41 PAGE MEMORY

Regular User ROM space
Watch for conflicts
Be careful here
HP41CL/CX fixed pages
Not available in HP41CL

TABLE 2 - HP41 PAGE/PORT MEMORY MAP LEGEND

In the classical HP41 you would plug physical modules containing applications in one of the four ports on the rear of the calculator. Most modules would map straight into the memory location designated to that port. Some modules (like the printer, timer or HP-IL module) can be plugged in any physical port (Ports 1-4 in the map above), but are hard-wired to take a pre-designated place in the memory map. The HP41CL allows physical modules to be plugged, like in your classic HP41, and at the same time plug virtual modules, or ROM images, into almost any place in the HP41 memory map. And here some caution is needed as potential conflicts can happen, resulting in a lock-up or other unexpected results. The Port concept (including Upper and Lower Ports) is still used in the 41CL as you will see. Due to its physical location the card reader will always be in Page E.

FIGURE 2 HP41 PHYSICAL PORT MAP

In the diagram above one single port has two 4K pages. Most simple plug-in modules (MATH for example) have only one such page and typically take the lower addressed part of the port. Some modules, like the famous PPC module, use both pages in one port. These are 8K modules. To complicate matters, each page has the possibility of having 3 additional shadow pages, or Banks. Code which is in a bank only shows up after executing a special instruction, this is called Bank Switching. For the more casual user this is fully automatic, just keep in mind that there are modules (HEPAX being the most famous one) that occupy only one Page, but have 4 times the capacity

of one Page. Multi-bank ROMs are used heavily in more advanced configurations, and the 41CX uses bank switching as standard for the additional Extended Functions.

The HP41 system has a second separate memory space, which is the User Memory space. This is where the stack, user programs, status registers (flags for example), user registers and extended memory are. Also your key assignments are stored here.

HP41 Address	Name	Used for
0x301 – 0x3EF	X-Memory	X-memory data
0x201 – 0x2EF	X-Memory	X-memory data
0x0C0 – 0x1FF	User Memory	User Registers, Programs, Key Assignments, Alarms, Buffers
0x040 – 0x0BF	X-Memory	X-memory data
[not used]		
0x000 – 0x00F	HP41 Status registers	User stack, ALPHA, flags, return stack, SIZE info, OS use

TABLE 3 - HP41 USER MEMORY

In the HP41 and HP41CL world we recognize different types of physical memory:

- ROM: Read Only Memory, this is a memory type you can only read from. Typically this used to be everything in the 64K HP41 memory space. A physical module is factory programmed ROM, and in many cases we use both the term ROM and Module for this. ROM memory keeps its contents when power is removed.
- RAM: Random Access Memory. Typically all the User Memory. Can be written to by the user (or a program) at the same speed as reading, and loses its contents when power is removed. In the HP41 this RAM is protected by the battery when switching the calculator OFF, which is why the user programs and register contents are still there when you switch the calculator on again. Can be corrupted when removing the battery or by misconfigurations, which leads to the **MEMORY LOST** message. HP41CL RAM is actually SRAM, the formal technical word used where the S is for Static. Due to its low power consumption its contents can be kept with a small battery, but battery removal or failure means that the contents are lost.
- QRAM: Quasi RAM. This has become the going acronym for something in the HP41 memory space that behaves like a ROM Module, but contains memory that can actually be written to within the Port memory space. This enables the user to write microcode programs. First used in MLDL (Machine Language Development Lab), later used in the HEPAX module to offer data storage in addition to creating User and microcode programs in the ROM page space. The microcode instruction to write to this memory was originally an unused instruction for the HP41 and decoded by MLDL type hardware.
- FLASH memory: this memory is used in the HP41CL hardware and typically behaves like ROM. Reading is very easy and the contents are kept when power is removed. Writing to FLASH is a bit more difficult since only a ‘1’ can be changed to a ‘0’ by a write operation. Older MLDL systems used EPROM, a ROM type which can be erased and programmed by the user in special equipment. Writing and erasing FLASH memory can be done in the HP41CL and requires special precautions and a strict programming sequence, which are taken care of by several HP41CL functions.

8. Memory on the 41CL

The HP41CL has physical memory of two types: FLASH and RAM. The HP41 ROM memory space is mapped into this physical memory using a Memory Management Unit (MMU). Any access to a location in the HP41 memory

map, which could be from a running program or request for data in a module, passes through the MMU. The MMU then decides where in the physical memory the access should be passed on. The MMU can be seen as a table with references from a Page to a location in physical memory where the contents or a ROM module resides, which could be FLASH or SRAM. Besides the mechanism controlling memory access and the MMU, the easiest way to look at it is as a simple table. And this table actually exists in the HP41CL physical memory map. HP41 User Memory is mapped in SRAM.

For example, when your user program wants to access a virtual module in the lower page of Port 2, Page A, the HP41CL processor looks at the entry in the MMU belonging to that page, and there it finds where in physical memory the actual module contents are stored. And by changing the MMU contents in the correct way you can change the contents of a Page, the virtual equivalent of unplugging one module and then plugging another.

Important to understand is that this MMU can be enabled and disabled. When disabled only a minimum number of address translations are made to give you a basic working HP41CL, ready to be configured. When enabled your configuration becomes active. Or not, when there are conflicts.

Module contents are typically referred to as ROM images or ROMs. This is simply a copy of the contents of a physical module, or code that never even existed as a physical module, or that you have made yourself. I will not go into much more details about this and the difference between microcode (or mcode) and User Code (FOCAL). Best read the HEPAX manual for this, or one of the older books (see references).

The contents of FLASH memory are kept when the calculator is powered off or the battery is removed. RAM (including the MMU table) can be erased when the battery is removed. A **MEMORY LOST** condition affects only the HP41 User Memory portion, and normally does not affect the MMU. But the MMU is disabled after a **MEMORY LOST**, so you need to enable it to get your configuration back. The exact organization of the memory as seen from the HP41CL processor will be handled later as we do not need it now for most of the basic functions. The HP41CL comes in versions with different memory amounts referred to as V2, V3/4 or V5 HP41CL hardware.

Page	Address	Contents
0x3E0 – 0x3FF	0x3E0000 – 0x3FFFFF	User reserved FLASH (32 pages)
0x2E5 – 0x3FF	0x2E5000 – 0x3FFFFF	Empty Flash pages v5 (as of January 2022)
0x200 – 0x2E6	0x200000 – 0x2E6FFF	Additional ROM Images v5 FLASH
0x1F8 – 0x1FF	0x1F8000 – 0x1FFFFF	User available FLASH v3/v4
0x100 – 0x1F7	0x100000 – 0x1F8FFF	Additional ROM Images v3/v4 FLASH
0x000 – 0x0FF	0x000000 – 0x0FFFFF	ROM Images in v2 FLASH (smaller FLASH)

TABLE 4 - HP41CL FLASH MEMORY MAP

Page	Address	Contents
0x818 – 0x87F	0x818000 – 0x87FFFF	Available
0x810 – 0x817	0x810000 – 0x817FFF	Sector Buffer Page 0-7
0x80D – 0x80F	0x80D000 – 0x80FFFF	Available (3 pages)
0x80C	0x80C000 – 0x80CFFF	YFNC, reserved for custom
0x807 – 0x80B	0x807000 – 0x80BFFF	Available (5 pages)
0x806	0x806000 – 0x806FFF	CFDB, Memory Buffer or CFLDB
0x805	0x805000 – 0x805FFF	YFBR, 41CL Memory Buffer, IMDB copy
0x804	0x804000 – 0x804FFF	HP41CL MMU registers
0x801 – 0x803	0x801000 – 0x803FFF	Unused HP41 Registers 400-FFF (Expanded Memory)
0x800 – 0x800	0x800000 – 0x800FFF	HP41CX Registers (incl X-Memory) 0-3FF

TABLE 5 - HP41CL SRAM MEMORY MAP

9. What is in my 41CL

The FLASH memory on the 41CL contains nearly all available known ROM images. A very detailed list is in the file mem_ref.pdf, that can be downloaded from the 41CL website. Some ROM images, and this file as well, are regularly updated when the images change or new images are added. All the ROMs in FLASH are identified by a 4-character unique ID, and these play an important role. The ID's are listed in the file mem_ref.pdf. Keep this file near you as we will use it frequently, and we will use the ID's all the time.

When first starting a virgin 41CL, new or after battery removal, it shows the dreaded MEMORY LOST and it almost looks like a normal HP41CX. This is because the MMU is disabled. You may or may not have the TIME module installed, but even without the physical TIME module or the new optional Time Module clone you will see this module in a **CAT 2** listing. But the devil is in the detail. There is one specific HP41CL ROM in port 7: -YFNZ 4G, the 41CL Extra Functions. This conflicts with the HP-IL physical ROM module (which also contains the IL Printer ROM), and the HP-IL module cannot be used until the Extra functions ROM is moved. We will do that later.

You also get a few extra functions in the Extended Functions Module (normally built in the HP41CX) that are not directly relevant to the 41CL, but very useful. Then there are some extra TIME functions under the header -CL TIME.

10. Getting your hands dirty

From this point we are assuming a new and empty HP41CL showing **MEMORY LOST** after battery removal. With no physical modules plugged with the exception of the TIME module (if you have that).

Step 1: Do a **CAT 2**, and check if you see the regular 41CL contents, including the -YFNZ 4G module and the 41CL extras. We will use it's ID ***YFNZ*** from now on.

Step 2: **XEQ "MMUCLR"** (a function inside ***YFNZ***)
This clears any rubbish from the MMU and initializes to a clean HP41CX configuration. SRAM powers up with random contents that will create havoc when not properly initialized.

Step 3: **XEQ "TURBO50"** (also inside ***YFNZ***)
Now do a CAT 2 again, or write a small user program, and enjoy your new superspeeded HP41. Not certain what the current speed setting is? Simple **XEQ "TURBO?"** and the setting is returned to X. Keep it at 50, no need now to slow down, unless you are running out of battery power.

Step 4: **XEQ "MMUEN"** (again, in the ***YFNZ*** ROM)
This enables the MMU and gives you a clean HP41CX, so do a **CAT 2** again.

Step 5: **XEQ "MMUDIS"** (the next function of ***YFNZ*** you will need)
This disables the MMU and normally gets you back to the port configuration in Steps 1-3, but not now!

HELP! Did you see **NONEXISTENT**? Well done, by enabling the (practically empty) MMU you also lose the entry for the ***YFNZ*** rom in Page 7. The MMU now only contains entries to refer to the basic HP41CX ROMS, and anything else is gone.
Now without ***YFNZ***, there is no way to do any manipulation of the 41CL MMU, and we cannot plug any virtual modules from FLASH and we have painted ourselves in a corner. We cannot even disable or enable the MMU.

Fortunately we have a life saver. The original HP41CX ROM is slightly modified and contains a special function that is always available (actually, it is in the TIME module).

Step 6: **XEQ "YRES"** (this is in the 41CX ROM)
This disables the MMU and always gets you back to the configuration in Step 1-3. Verify by doing **CAT 2**.

In case you did not notice, stack, TURBO mode and user memory (all your programs) are preserved when playing with the MMU. Please remember this command, you might need it!

It is very well possible, by not taking care of interactions between plugged ROMs, that your calculator becomes unresponsive. In that case the memory lost sequence (**[backspace]** and **[ON]**) may work, otherwise remove the battery for a few seconds (or longer, this could be up to a few minutes) to enforce a **MEMORY LOST**.

Step 6: Enter **"YFNZ"** in ALPHA, then **XEQ "PLUG1L"**
This command sequence plugs the ***YFNZ*** module, with our basic HP41CL Management functions, in the lower Page of Port 1, in Page 8. Again, make certain that no physical modules are plugged. Refer to the manual of the 41CL Extra Functions (the full name of ***YFNZ***) for the other functions, in case you wish to plug this module in any other port.

Step 7: **XEQ "MMUEN"**
Now the MMU is enabled. Do a CAT 2 and verify that you see ***YFNZ***. You have just enabled a fully working custom configured HP41CL

Step 8: **XEQ "YFNS?"** (another useful function in ***YFNZ***)
In many configurations it is difficult to find out where you plugged ***YFNZ***. This handy function returns the page it is plugged in X. You should see 8 in the X-register. This prevents accidental overwriting of ***YFNZ*** and having to do a **YRES**. The HP41CL is ruthless in this regard. If you would now do another **PLUG1L** of any module then ***YFNZ*** is gone from Page 8.

Now comes the real fun, we will plug another module in the 41CL. You can use any module you like, for this first example I prefer to keep things simple and use pretty straightforward ROM images so we do not run into complications like bank switching.

One of the very first real modules (or Application Pacs as HP called it) I owned was the MATH module, so let's plug that first. Use the earlier mentioned memory reference list and find the MATH module. The HP41CL system is a really smart design, and by using the ID's we do not have to worry about the exact FLASH location of any ROM image.

Step 9: enter **"MATH"** in ALPHA, and **XEQ "PLUG1U"**
Do a **CAT 2** and verify if you see the good old MATH module, and ***MATH*** is the ID of this module.
We have actually done something pretty special. ***MATH*** is now plugged in the Upper Page of Port 1. The physical module, when plugged in Port 1, would take the lower Page. ***MATH*** now sits in Page 9!
Be careful, the **PLUG** functions do not check if there is anything already plugged in that page, existing MMU entries will be overwritten. The content of the previous module will not be overwritten, these are safe in FLASH memory..

One of my other favorites of the good old days was the PPC ROM. Loaded with many kinds of functions and using so-called synthetic programming (see references) was an amazing achievement of the PPC user group who put their own code in a custom physical ROM.

Step 10: enter **"PPCM"** in ALPHA and **XEQ "PLUG2"**
This plugs the PPC module (which has 2 pages) in Port 2, or Pages A and B. Verify again with a **CAT 2**.

Step 11: enter the following small program to find prime numbers

```
LBL PNG      (a Prime Number Generator)
2
LBL 01
XEQ "NP"     (the function from the PPC ROM)
X=Y?
VIEW X
CLX
2
ST + Y
GTO 01
```

Switch off turbo mode with **XEQ "TURBOX"**, type in an *odd* number and **XEQ "PNG"**

Interrupt the running program after some primes with **[R/S]** and go to full speed: **XEQ "TURBO50"** and run the prime number generator again. If you are interested feel free to try the various TURBO modes (**TURBO2**, **5**, **10**, **20** and **50**, see the 41CL manual)

You have to keep track of which ROM images are plugged where. And try not to plug something in the reserved pages (see Table 1 - HP41 Page memory) unless you really know what you are doing. There are more advanced functions available to check which ROM is in which page and we will come to those later.

The various **PLUG** functions may result in an error message when something goes wrong, please have a look at page 25 of the 41CL manual.

Step 12: **XEQ "UPLUG1U"**
This must be done with care, and unplugs anything in the upper Page of Port 1 (remember, our MATH module was there). With the ***YFNZ*** functions you cannot unplug by name. There is also a function **UPLUG1**, which unplugs anything from both pages in Port 1, and then you would lose ***YFNZ*** itself. Verify with a **CAT 2** that ***MATH*** is unplugged, and plug this in again.

Step 13: enter **"41AD"** in ALPHA and **XEQ "PLUG3"**

> ***41AD*** is the ID for the HP41 Advantage Pac, a pretty advanced collection of functions and programs for mathematics, including matrix functions. What is most special is that this is a bank switching module. It uses 2 Pages and contains 3 active pages. Verify this again with a **CAT 2**, the bank switching is invisible from here so you will only see the module itself.

The table below shows the current contents of your HP41:

Port	Page	Address	Bank 1	Bank 2	Bank 3	Bank 4
4	F	0xF000-0xFFFF	[Port 4, upper Page]			
	E	0xE000-0xEFFF	[Port 4, lower Page]			
3	D	0xD000-0xDFFF	ADVU1-1B	ADVU2-1B		
	C	0xC000-0xCFFF	ADVL1-1B	ADVL1-1B		
2	B	0xB000-0xBFFF	PPCU			
	A	0xA000-0xAFFF	PPCL			
1	9	0x9000-0x9FFF	MATH-1D			
	8	0x8000-0x8FFF	YFNZ			
	7	0x7000-0x7FFF				
	6	0x6000-0x6FFF				
	5	0x5000-0x5FFF	[TIME MODULE]	[CX FNS - bank 2]		
	4	0x4000-0x4FFF				
	3	0x3000-0x3FFF	[CX FNS – bank 1]			
	2	0x2000-0x2FFF	HP41CX OS – ROM2			
	1	0x1000-0x1FFF	HP41CX OS – ROM1			
	0	0x0000-0x0FFF	HP41CX OS – ROM0			

TABLE 6 - HP41 PORT OCCUPATION

Please feel free to experiment a bit more and plug your own favorite ROM image while taking a note of what is in each port. A possible source of conflict is the XROM number, and this number is listed in the mem_ref file. In general there should be no multiple modules with the same XROM number in your calculator. We will talk later about how to handle this issue when this is going to be the case in your configuration.

When I first started to work with my 41CL one of my thoughts was that it would be great to be able to save a configuration, and switch to another one. Luckily, this is totally possible with your 41CL. But not so easy in the setup with only YFNZ to handle the 41CL specifics. For that we first need to enhance our configuration.

And a disclaimer: all the above is valid for my V5 HP41CL board. This version has much more FLASH memory and contains more ROM images. If you have a Version 2 board, not all ROM images in the mem_ref list are in your system, then you need to refer to the correct mem_ref for your version. What to do if you want to plug a rom that is not in the list? We will come to that later.

11. Enhancing your 41CL

The ***YNFZ*** image is necessary to bootstrap your 41CL and perform some basic functions. There is however an enhanced version of this module called *41CL Extreme Functions* (the ID is ***YFNX***) and that has the basics of ***YFNZ*** plus several extra features that makes handling your 41CL a bit more user friendly. In addition, this module does a check for some possible conflicts and makes it a bit more difficult to overwrite its own MMU entry by accident. ***YNFX*** has a secret companion, this is ***YLIB***, or the Extreme Functions Library. It is secret because it dynamically

loaded when ***YFNX*** needs it and temporarily sits in Page 4, and is removed when the function is finished. Anything that was in Page 4 is then restored, and you do not have to plug ***YLIB*** in Page 4.

Care should be taken when using a configuration with HP-IL. This physical module is always located in Page 7, and the printer takes Page 6. Important to know is that the little switch on the IL module, which disables the printer, actually parks the Printer ROM in Page 4. With only ***YFNZ*** this is not very relevant because the internal ROM images take priority over physical modules, and ***YFNX*** does a check for such conflicts and gives priority to the physical module. But ***YFNX*** should **never** be used with the IL Printer disabled (or with any other physical module using Page 4)!

In the next examples we assume the configuration from the previous section with the MMU enabled

Step 1: enter **"YNFX"** in ALPHA and **XEQ "PLUG1L"**
This plugs the 41CL Extreme Functions in Page 8, overwriting ***YNFZ*** that was there before
Verify with a **CAT 2** that ***YFNX*** is indeed plugged. By doing **XEQ "YFNS?"** **8** is returned to X, which is indeed the targeted page.

Step 2: enter **"YFNZ"** in ALPHA, enter **8** in the X-register and **XEQ "PPLUG"**
We have attempted to plug the old ***YFNZ*** in the port where we have just put ***YFNX*** in. The result is a **"LOCK ERROR"** , since our ***YFNX*** module protects itself against accidental removal. Nice feature! In addition, you have now used the **PPLUG** function, which allows you to put the page number in X, and the ID in ALPHA, and plug a module in any port.

YFNX has so many nice functions that I would strongly recommend to go through the manual (see references). For now we will focus on some functions that you are likely to use every day, and we need to do a slightly deeper dive in the workings of the MMU.

The MMU is a table that translates an HP41 address in the Page Memory map to a physical address in FLASH or SRAM. Every entry in this table has a number of attributes, and one of these allows the *locking* of a page. This means that this MMU entry cannot be changed or cleared. The HP41 Systems ROMS (also known as the HP41 Operating System) are locked by default (with a different lock mechanism) to prevent turning you calculator into a brick. It is possible to unlock any page, and there are examples of alternative HP41 Operating Systems. In our current configuration we are pretty stuck with ***YFNX*** in Page 8, as ***YFNX*** will not let itself be unlocked nor moved to another page. Check the manual of ***YFNX*** for the functions **EXPG** and **MVPG** on how to move a module from one page to the other. Keep in mind again that moving a ROM into another Page does not move the ROM contents, it just changes the entry in the MMU for the respective Pages.

In case we want to move ***YFNX*** to another page we need to go back to a situation with the MMU disabled.

Step 3: **XEQ "MMUDIS"** and **XEQ "UPLUG1L"**
This unplugs ***YFNX***, test this by enabling the MMU again and do a **CAT 2**
The only way now to get ***YFNX*** back is to disable the MMU, and the only way we now have is with **YRES**!
So **XEQ "YRES"**, enter **"YFNX"** in ALPHA en **XEQ "PLUG1L"** followed by **XEQ "MMUEN"**. You could plug ***YFNX*** in another free page, but let's keep it in Page 8 for now.

The HP41CL supports multiple MMU copies, and there is always one active, which is the *Primary MMU*, and this is set number 0. The others are set number 1 to F (hexadecimal 15), these are the *secondary* MMU registers. Remember that the MMU can be disabled which gives you a clean HP41CX with ***YFNZ*** enabled. We have already switched between different HP41 configurations: a clean HP41 CX and one loaded with MATH and PPC ROM by

simply enabling and disabling the MMU. ***YFNX*** comes with a number of powerful (and dangerous) functions to manage the MMU register sets, so be careful here.

In the next examples we will play with some different MMU configurations. First of all be aware that the MMU register sets, with the exception of the primary MMU, are not initialized and may contain random information which will most likely mess up your calculator when used.

Step 4: **XEQ "MMUCLS"**
This clears all secondary MMU registers, and does not touch the primary (current) MMU registers, feel free to confirm that with a **CAT 2**

Step 5: **XEQ "STOCFG" 1** (enter 1 at the prompt)
We have now stored our current MMU setup in the secondary MMU register set #1

Step 6: **XEQ "MMUCLP"**
This clears the primary MMU registers, with the exception of those in locked pages. Do a CAT 2 to confirm: all plug-in module are gone, only ***YFNX*** remains because that was locked (and of course the standard HP41CX modules are still there)

Step 7: **XEQ "RCLCFG" 1** (enter 1 at the prompt)
This function returns with a mysterious message 678. This restores the configuration saved in step 5, you will now have your configuration back, verify this with **CAT 2**.

Be aware that pages that were locked will not be restored. The message that was returned actually means that pages 6, 7 and 8 were locked and not restored. In your specific case this may have been a bit different, in my case I usually have HP-IL plugged in Page 6 and 7, Pages with physical modules are locked by ***YFNX***. Want to check if a page is locked? Then use the function **LOCK?** And type the page number (for A -F you need to go to ALPHA mode first on the prompt).

There is one nice little trick here: You can have an alternate configuration with ***YFNX*** in a different Page. In this case ***YFNX*** allows itself to be moved (a feature available since ***YLIB*** version -4D). Another trick: **RCLCFG 0** will list the locked pages, handy to check this before changing configurations.

The MMU configurations will survive a **MEMORY LOST**, but will be gone when the battery is removed (although a few seconds will usually be ok). We will come back later to some more tips and tricks around the MMU register sets.

Now let us make a final step in enhancing our HP41 configuration. Back in the good old days I always had the CCD ROM loaded. This had really great extensions to the operating system, of which the expanded CAT function was my most popular. After some iterations the mcode wizard Ángel Martin has built upon the concept of the CCD module and brought it to new heights with his AMC-OSX module. It contains many original contributions plus a compilation of routines from others. I personally think that no HP41 calculator should go without it. The ID for this module, from mem_ref, that we will use from now on is ***OSX3***.

OSX3 is a ROM image that sits in only one page and uses 4 banks, so the full capacity of the HP41 is used. In addition, a second module is needed, and this is Ángel's Library-4 module. This is a very special library that is used by a number of Ángel's other modules and sits in Page 4 and is a requirement to have loaded for ***OSX3***. Please remember that you will now have a Page 4 module active, and this should *not* be used in combination with a disabled HP-IL Printer module! In the next step we will install Library 4 (ID: ***4LIB***) and ***OSX3***.

First we need to make one important decision: where to plug ***OSX3***. In my ideal world the physical user ports should always be more or less freely configurable for the user. When ***OSX3*** is always in my configuration I prefer to plug that in port 6 or 7, but this would conflict with an HP-IL module (of which the printer must always remain enabled due to the Page 4 conflict). The non-IL printer (and the serial printer ***YPRT***) only takes port 6, so my port of preference for ***OSX3*** is Port 7. Please disagree and make your own choice, in the next examples Port 7 will be used.

And a word of caution: Page 7 is relatively safe to plug in most modules. Port 6 (normally reserved for the printer) is *not* safe for casual use. See page 26 of the HP41CL manual for a list of modules that can be safely plugged in Page 6. This list includes HEPAX, ***OSX3*** and ***YFNX*** plus some others.

I would normally plug ***YFNX*** in Page 6 in this configuration, but unfortunately this will interfere with some of the ***OSX3*** functions, and ***YFNX*** should be in a higher numbered Page above ***OSX3***. Now we will first plug ***YFNX*** all the way up in Port F.

Step 8: **XEQ "YRES"** This disables the MMU and gets you back to a clean HP41.
XEQ "UPLUG1L" Unplug ***YFNX***
Enter **"YFNX"** in ALPHA and **XEQ "PLUG4U"** to plug ***YFNX*** in Page F
XEQ "MMUEN"
Verify you port configuration with **CAT 2**

Step 9: **XEQ "PLUG" "OSX3" 7**
Now toggle the ON switch to restart your HP41CL. You will now get the message **NO LIBRARY**, indicating that ***OSX3*** is missing its good friend Library 4. Do not try to do anything else yet, although no harm will be done. This again shows how smart the PLUG function is, as all 4 banks of ***OSX3*** are automatically loaded.

Step 10: **XEQ "PLUG" "4LIB" 4**
You have now plugged something in Page 4! You may now **LOCK** Page 4 to prevent ***LIB4*** from unintended unplugging

You now have many many tools at your disposal to make your life with your calculator much easier, and things are a bit different. Most changes that you will notice immediately:

- ALPHA keyboard: in USER mode this is the same as you know already. With **USER** mode OFF you can type in lower case characters. This may be a bit confusing at first (most lowercase characters display as a starburst), but you will get used to this very quickly
- The **CAT** function looks different, you will see **CAT"** (with the quotes) to remind you that this is now different:
    - **CAT"1** and **CAT"2** are not different at first sight
    - **CAT"3** prompts for a number and will list the internal functions starting at that number
    - When you stop a **CAT** with **R/S**, the **SHIFT** is sticky with **BST** and **SST**
    - Stopping a **CAT** and pressing **XEQ** at a certain function will execute that function.
    - **CAT"7-F** will list the catalog of that Page

- **CAT"0** will give you a choice. Very useful is option G, which will list all Pages in the calculator. You now have a way of knowing what is plugged in which port. There is a function PGCAT in ***OSX3*** which does the same, but typing **CAT"0 G** is much quicker. Note that you cannot R/S and SST this catalog, pressing **[ENTER]** will hold the listing.

The **PLUG** function in ***YFNX*** is extremely powerful, and there are some tricks that are useful to know. First be aware that **PLUG** used a 'sticky' ALPHA mode. When **PLUG** shows its prompt, ALPHA is OFF and you can type numbers, but ALPHA has to be toggled to be able to type characters. When ALPHA is on, you can type numbers with **[SHIFT] [number]**. When you have typed 4 characters it immediately jumps to the prompt for the Page number, but you can still backspace to the ID. And the Page number can be a digit or a character (A-F), so you may need to toggle ALPHA again.

Some tricks in using **PLUG** (try it!):

- **XEQ "PLUG" [ALPHA] [ALPHA]**: enters ***EMPT*** as the ID, this will unplug anything that is in the selected page
- **XEQ "PLUG" "????" ?** will scan all Pages and return a list of plugged ROMs with their ID. Actually only the first character in the ID can be a question mark, typing four times ? is just quicker. This takes some time because it does a search through the HP41CL's internal database of ID's. This database will be explained in more detail shortly as it is a very important part of the HP41CL architecture. One word of caution here: the HP41 OS pages are not listed correctly with this **PLUG** command.

We have now achieved many things and your HP41CL is much enhanced. You should now be able to create, save and restore some configurations, and I hope you like ***OSX3***. We will use ***OSX3*** for some other tricks. In the next sections ***OSX3*** (and ***4LIB*** of course) will always be in the calculator, and ***YFNX*** is moved to Page F. Moving ***YFNX*** wil

Get rid of any pages you do not like, or save you configuration and get ready to move on. In the next examples we will assume an HP41CL with only ***OSX3*** in Page 7 and ***YFNX*** in Page F, so unplug whatever is in the other ports.

12. Becoming a 41CL poweruser: more complex tasks

FIGURE 3 MY HP41CL

We have already used the ID's of ROM images and we have seen that we can automatically plug in multi paged or multi banked modules. We have to thank Monte, the designer of the HP41CL, for setting up a system not only with excellent hardware, but also with a wonderful structure of tools and architecture.

Central in the HP41CL is the ***IMDB***, or Image DataBase. This is where the ID's are stored, along with characteristics of the module. Part of the mem_ref file mentioned earlier is an extract of this Image DataBase. If you take a close look at the mem_ref file, you may notice that this database has its own ID: ***IMDB*** and actually looks like a module, although it cannot be plugged in. ***IMDB*** has a companion which is ***FLDB***: Flash YCRC Database. When you **PLUG** a module, the ROM ID will be searched for along with the characteristics of the module and used for programming the MMU entries according to the needed number of Pages and Banks. We will now take a deeper dive in the HP41CL memory structure. The mem_ref file actually contains a very accurate description of the 41CL memory.

We have learned earlier that the 41CL has two memory types: FLASH and SRAM. All modules in the IMDB are in FLASH in its shipping configuration, including ***IMDB*** itself. FLASH memory is divided into pages of 4K (4096) words (a word is defined here as 16 bits), and one such page (not accidentally) matches one ROM module image. The pages have hexadecimal addresses, and Page 0x000 contains the first page of the HP41CX Operating System, and ***YFNX*** is in Page 0x00A. A version 5 PCB of the HP41CL can keep 1024 Pages. In the current version 740 pages are used (some pages are empty or reserved), the rest can be used. The highest numbered page in FLASH is at 0x3FF.

For some reason I have a personal preference when I make a picture of a memory model to have the lowest address at the bottom. I will stick with this habit, but the mem_ref file does this the other way around.

The HP41CL SRAM has the same page architecture as FLASH, the v5 board of the HP41CL has a total of 128 pages in the address range 0x800 to 0x87F, roughly split in 3 main functions:

- HP41 User and System registers (stack, programs, Extended Memory, User Registers)
- 41CL MMU registers (secondary and primary)
- Some reserved buffer space (which we will use!)
- Sector Buffer (used when programming FLASH memory)
- Loads of free space for us to use

So far we have been able to avoid the exact addressing within the memory map. When using SRAM this now becomes different and we need to be able to get into the details here.

Maybe you ask yourself: why use SRAM at all? If you are happy with the way your HP41CL operates until here, then there is no need to move on beyond this point. But SRAM has some unique features to offer:

- Manipulation (backup or different sets) of the User Registers and Extended Memory
- Modifying existing ROM images, for example in case of an XROM numbering conflict
- Using one of the jewels of the HP41 world: HEPAX
- Using a ROM image that is not available in the ***IMDB***
- Creating your own ROM image with unique functions (in either mcode or Focal, or a both)

The last option may sound like something for mcode nerds, but really it is not. HEPAX allows you to move your collection of User Programs (do you have these written down, printed on fading listings or still on magnetic cards?) written in Focal into a ROM image structure and then make this ROM image part of the IMDB, even in FLASH memory if you like so these programs are saved forever and available with a few keypresses.

As a first step we will install HEPAX in our calculator. HEPAX was sometimes called the holy grail of the HP41. It was introduced by some very smart Danish programmers and consisted of a single Page, 4-bank module and came

with the option to have an extra RAM module installed with up to 4 pages of SRAM that simulated ROM memory. This is now also called QRAM (for Quasi RAM) and of that we have a lot in our HP41CL. Unfortunately the original HEPAX hardware module was introduced quite late in the HP41 life cycle and was not a great commercial success. As a result HEPAX modules were very hard to find when the HP41 was taken off the market and became a collector's item. With devices simulating ROM modules (like my own designed MLDL2000, Clonix and NoVRAM and of course the HP41CL) becoming available the HEPAX module was within reach of many users. There were complications as HEPAX had some hardware dependent peculiarities that were very difficult to implement in simulated ROM (or QRAM). This was overcome by patching HEPAX and some special instructions to initialize the HEPAX dependent RAM modules.

Remember that for the next steps we only have ***OSX3*** and ***YFNX*** loaded in our calculator (respectively in Pages 7 and F)!

Step 1: **XEQ "PLUG" "HEP2" 8**

This command will plug HEPAX (all four banks of it) in Page 8. ***HEP2*** is the ID for the patched HEPAX and that is important. Verify by doing a **CAT" 0 G** and make sure this is HEPAX version 4H (or higher). Then do an OFF/ON cycle. This is important to initialize HEPAX itself.
Next do a **CAT"0 H**. This executes **HEPDIR**, or the directory of the HEPAX managed RAM modules or File System. Currently we have none, so the function responds with **H:NO FILESYS** as this is not yet installed, and that will be our next task.

Installing the HEPAX File System requires some careful planning:

- Do you want 1, 2 or 4 pages of HEPAX RAM (HEPRAM) in the HEPAX filesystem? In the next example we will use 2 pages. This is a good compromise between Pages used and space available in the HEPAX filesystem. Keep in mind that these Pages need to be consecutive, there should be no gaps between them. This is not entirely true, but this will complicate matters unnecessary. When you need more space in the HEPAX file system you can actually expand it, but only upwards into the next immediate adjacent Page.
- These pages will be in SRAM, so we need to decide where in SRAM to place them, and avoid used or reserved space. Assuming that we have nothing in user available SRAM, we will use the SRAM space in the 41CL memory map in page location `0x808` to `0x80B`, with room for 4 RAM pages.
- The HEPAX File System must be initialized in a very specific way and we need to understand how. The original HEPAX actually does this automatically, but the hardware features to do this are not available on our 41CL.

When we plug in a page which is SRAM we must ensure that the contents are initialized, otherwise the HP41 may lock up due to unexpected instructions. Fortunately there is an example of an initialized SRAM page prepared in FLASH. This cannot be simply plugged in a Page we need to do a bit more.

Step 2: Enter in ALPHA: **"0B9>808"** then **XEQ "YMCPY"**
Edit the text in ALPHA to **"0B9>809"** and **XEQ "YMCPY"**

What happens here is that we copy a block from FLASH (which is our HEPRAM page empty template) into the SRAM page `0x808`. Check the mem_ref file for the ID ***HEPR***, this is our HEPAX SRAM template, and located at FLASH page `0x0B9`, this is the template we have now copied to SRAM, and we copied the same block to SRAM page `0x809`. **YMCPY** is a function in ***YFNX*** that copies an entire 4K block and ideal for this purpose. It automatically switches the speed of the 41CL to turbo mode 50 (maximum speed) while copying.

These copied pages in SRAM are not visible with a CATALOG function, we first need to plug these in a port by telling the MMU to point to those pages in SRAM, and we need to decide in which Pages we will plug the HEPAX RAM (or HEPRAM). Keep in mind that these need to be in adjacent pages. We have HEPAX in Page 8, and I want to keep Page 9 free for now (we will see later why), so let's plug the HEPRAM in Page A and B

We do that with the following command.

Step 3: **XEQ "PLUG"** and enter in the prompt: **"-808"** (first character is a minus) and **"A"** at the second prompt
XEQ "PLUG" "-809" "B"
Now we have plugged our HEPRAM templates in Page A and B. Do a **CAT"0 G** and look at what you see in Page A and B: it says **NO FAT** instead of **NO ROM**.

The message **NO FAT** means that the CAT function did find something in the page, but not very much. FAT means Function Address Table, a list of the functions in a ROM, and clearly our ROM image does not have any functions. But the message means that we have successfully plugged the HEPRAM templates in our calculator.

The next step is very complicated if we would have to do that manually. HEPAX uses a smart but complicated scheme to connect its HEPRAM pages in a chain, and this chain must be initialized. HEPAX does some initialization by itself when it powers up and this can be shown by doing the command **HEPDIR** (HEPAX Directory) or in the short version **CAT"0 H**. It will show **DIR EMPTY**, and leaves the amount of free registers in X. This is **651**, which is the free space in the first HEPRAM module. To do the final initialization we need some help, and again Ángel Martin has created another powerful tool which is the POWERCL EXTREME module. This ROM image contains another collection of functions with some overlap with ***OSX3*** and ***YFNX*** and is another 4-bank beauty with the ID ***PWRX*** that we now need for just one function to initialize our HEPRAM. It takes one page only and we will load it in Page 9. ***PWRX*** actually has functions that will do the full preparation of pages as we did in Step 3 above by hand. I deliberately wanted to let you do this manually to get a feel for what exactly is going on. Preparing the HEPRAM chain is not so much fun, so we use a little tool for that. Should you later wish to change your HEPRAM configuration, feel free to use the **HPX4**, **HPX8** or **HPX16** functions in ***PWRX***.

Step 4: **XEQ "PLUG" "PWRX" 9**
PWRX is now plugged in Page 9 and we are ready to initialize HEPRAM

Step 5: **XEQ "HEPINI" 2 "A"**
Note that a HEX keyboard is now enabled in the prompt. The first number we typed is the number of pages to initialize, the second is the Page location of the first HEPRAM page. There is some error checking in the function, but it is very well possible to fool the function and overwrite something unintended, so be careful here and keep track of the contents of your pages.
Confirm, by typing **CAT"0 H**, you will still see **DIR EMPTY**, but in X is now a larger amount of registers, this should be **1304** for two empty HEPRAM pages.
HEPINI can be used whenever the chain between HEPRAM modules is broken, and should leave the contents intact, please refer to the excellent ***PWRX*** manual for all the details.

In case you still have your **PNG** program from earlier exercises in User Program Memory, you could actually try to load in into HEPRAM as an exercise. Otherwise type another small program, enter its name in ALPHA instead of PNG in the next example.

Step 6: Enter in ALPHA **"PNG"** (or the name of your own program) and **XEQ "HSAVEP"**
This copies (actually does much more) your program to HEPRAM. Now do a **HEPDIR** or **CAT"0 H** and you will see your program in the listing. Now also do a **CAT"0 G**, and you will see that for Page A there name of your program is actually the ROM name! The reason is that the first function name in a ROM is taken as the ROM's name in a CAT listing.

If you now delete the program from user memory you can still run it (not **PNG** in this case as it depend on the PPC ROM which is now unplugged).

PWRX has goodies for managing HEPAX and many other aspects of your HP41CL. Feel free to explore it, but for now we will unplug it.

Step 7: **XEQ "PLUG" "EMPT" 9**
PWRX is now unplugged but not forgotten. In the example above you can literally type **"EMPT"** or use the **[ALPHA]** key twice to unplug a module from a Page.

The configuration of the MMU with HEPAX and HEPRAM can now be saved in one of the secondary MMU configurations. But be careful when restoring, as the HEPRAM pages must still be present at the RAM locations `0x808` and `0x809`!

Now there is one complication we need to get out of the way. There is no immediate action needed, but when your configuration changes things may be different. We have earlier mentioned that the XROM numbers as a possible source of conflict, and when you want to mix several modules of your choice there is a realistic chance that this conflict may occur. And such a conflict may go unnoticed in some cases until you are at a point that correcting this is a lot of work.

The XROM number is an identifier used by all plug-in modules. Due to limitations in the HP41 system setup this number can be only between 1 and 31. Have a look at the mem_ref file, there is a column XROM with exactly that number. ROM's without an XROM number are usually Page 4 ROM's and of course the HP41 Operating System. Keep in mind that your current HP41CL configuration always has XROM's 25 and 26 in use (Extended Functions and Time Module). When ***YFNX*** is plugged, it will do a check on startup of the calculator, and complain if it does not like the XROM configuration.

Step 7: **XEQ "CHKXROM"** (from YFNX or **XEQ "CHKSYS"** (from ***OSX3***)
This should tell you that your configuration is OK (I hope) and that there are no XROM conflicts

Step 8: **XEQ CAT"0 R**
This is an alias for the ***OSX3*** function **ROMLST**, and it returns the XROM's that are now in use in your system, so let's have a look at these. The list is put in ALPHA, and looks like this (yours should be the same if you followed all steps exactly):
"25:35:26:05:07:10:11:15" which means the following:

Page 3: XROM 25: Extended Functions
Page 5: XROM 26: Time module
Page 7: XROM 05: OSX3
Page A: XROM 10: HEPRAM Page A (decimal 10)
Page B: XROM 11: HEPRAM Page B (decimal 11)
Page F: XROM 15: YFNX

Take another look at the list in ALPHA, and there is an entry 35. This is actually ***4LIB*** in Page 4. It does not have an XROM number, but `0x023` (decimal 35) is a dummy number (but with an intention, this is a relatively harmless instruction for the HP41 processor) that happens to be in the first location in the ***4LIB*** ROM image, where the XROM number is stored. The **ROMLST** function only displays 8 XROM numbers (due to the maximum length of the APLHA register) so take care if you have many ROMs plugged, as you might miss one in this list.

The XROM numbers for the HEPRAM modules are actually the Page numbers (this is the default when running **HEPINI**), and this gives you an idea of possible conflicts. So you really have to plan ahead and consider what modules you want to plug in combination with HEPRAM. The big questions is of course what to do when there really are XROM conflicts, and what the consequences are. Normally you type in a command using **XEQ** followed by the name of the function, and in many cases an XROM conflict does not really matter, the HP41 will search for the function by name. Things become different when you use the function or program in a ROM, for example a program in HEPRAM. So let's do an exercise to demonstrate this, and practice changing the XROM number.

Step 9: Type in the following program in your calculator.

```
01   LBL "DICE"
02   LBL 00
03   XEQ "RNDM"        this is a function in OSX3, generating a random number between 0 and 1
04   6
05   *
06   1
07   +
08   INT
09   VIEW X
10   PSE
11   GTO 00
12   END
```

This is a simple program that throws a dice and shows a random integer ranging from 1 to 6, go ahead and test it.

Step 10: enter **"DICE"** in ALPHA and **XEQ "HSAVEP"**
The **"DICE"** program is now saved in HEPRAM, do a **CAT"0 H** to confirm Step

Step 11: **XEQ "CLP"** and enter **"DICE"** at the prompt
Your program DICE is now removed from user memory, you can confirm this with **a CAT"1**
But our program is now in HEPRAM, and since this behaves like a real plug-in module we can execute from ROM

Step 12: **ASN "DICE"** [to your favorite key] to connect the **DICE** program with a key
Make sure that USER mode is on and press and hold that key, you will see that the **DICE** program is now assigned to that key

Our next big step is to change the XROM number. This can be done in two ways, so let's take the easy way to change it, and the hard way to change it back. The easy way is using one of the tools in ***OSX3*** by using an editor to change the contents of a ROM image. This image must be in SRAM of course, do not try this on an image in FLASH! ***OSX3*** has a really nice hexadecimal editor called **HEXED** to view anything which is in ROM, and to edit QRAM contents. It will be able to change the content of ROM images that are plugged in a Page and which are in SRAM.

Please be very careful in following the next steps exactly! The keyboard behaves a bit different while in HEXED and is a full hexadecimal keyboard.

Step 13: **XEQ "HEXED"** and type **A000**:
Result: **A000 00A _ _ _**
This is the first word in Page A (where our first HEPRAM page is), and the value of that word is `0x00A`, or decimal 10, and this is exactly our XROM number.
If you type **[backspace]** there will be a new prompt for the address, type **B000** and you will see the XROM number of the second HEPRAM page, with XROM number `0x00B`, or decimal 11. Now use **[backspace]** or **[BST]** to get back to address `0xA000`
Let's use **00C** as the new XROM number (this is not in use in our current configuration) and type that in at address A000. **HEXED** will immediately go the next address A001, which is the number of functions in that ROM. With BST you can move back one address, confirming that the XROM number has indeed changed to 00C.
Get out of **HEXED** by pressing **[backspace]** twice, and press and hold the key to which you have assigned the DICE program. This will now show **XROM 10,**01 (or **XROM 10,00**). ROM functions are stored in key assignments and also in user programs with a reference to the original XROM number and index in that ROM. This has now changed, and the HP41 system has lost XROM 10. But the **DICE** program is still there, try it by doing **XEQ "DICE".**

If you would now plug another ROM with XROM number 10 (the original HP GAMES pac has this) then you would see the key assignment to that function index in the other ROM! So changing the XROM number after you have made key assignments or referred to the function in a user program will mess things up and it might be a lot of work to get everything sorted out again. Feel free to verify by doing **CAT"0 R** (gives the list of XROM numbers in ALPHA) and **CAT"0 H** (**HEPDIR**) to verify that the XROM number has indeed changed, and that the HEPRAM chain is still intact, despite the different XROM number.

We will now change the XROM number back to its original value, but the hard way. Remember that **HEXED** only works on a ROM that is plugged in a Page. We can actually access non-plugged memory locations directly in the HP41CL memory map with some other tools. This is a bit more complex because we need to know the exact memory location of the word we want to change, in this case the XROM number. In step 2 of this section we have copied the first HEPRAM template to HP41CL page `0x808` in SRAM, and this is the block of which we want to change the XROM number. Since the XROM number is the first word in a ROM image, the exact address would be `0x808000`.

Step 14: enter in **ALPHA "808000-0000"** and **XEQ "YPEEK"**
YPEEK is a function in ***YFNX*** that will directly read a memory location from the HP41CL memory map. This is independent from any plugged modules, and you can use this function to examine any location in FLASH or SRAM. A bit more friendly way of looking at memory contents is the function **YEDIT** in ***PWRX***, but let's do this the hard way now.
The last 4 characters in ALPHA are a placeholder, as the results of the read operation will be placed in ALPHA as you will see, and this is our XROM number `0x000C` (don't mind the 4 digit hexadecimal number).

Step 15: change the value in ALPHA to **"808000-000A"** (this value must be exact!) and **XEQ "YPOKE"**
YPOKE (again in ***YFNX***) will write the value in the last 4 digits in ALPHA to the memory location, and we have now restored the XROM number. Verify this with **CAT"0 R**, and by pressing and holding your assigned key. Your **DICE** assignment is now back!

Changing XROM numbers is not a casual process. Especially a ROM image with User programs (like the PPC ROM) cannot have the XROM number easily changed, as the programs in the ROM usually call other programs in the same ROM by referencing to its XROM number. ***YFNX*** checks the XROM numbers for a conflict upon startup and reports a CONFIG BAD when there is an XROM conflict, although many functions in the ROMs will work, just do not use the functions in one of the ROMs in a program or key assignments.

With access to HEPAX, HEPRAM and the ability to change XROM numbers you now have the ability to do almost anything with your HP41: plugging and unplugging modules, storing and recalling complete configurations and creating a HEPRAM full of your own user programs that will survive the dreaded **MEMORY LOST**. Please do remember that removing the batteries for a longer period will erase the SRAM contents.

In the next section we will move on to a next level, and that is the creation of your own ROM module (or the modification of an existing one) and adding that to the ROM Image database ***IMDB***. As an extra we will use some basic functions to communicate with another computer (or another HP41CL!).

13. Even more power: creating your own ROM

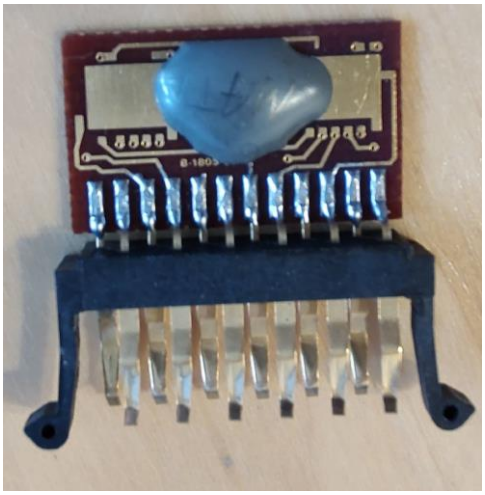

FIGURE 4 INSIDE A MODULE

Creating your own ROM image is one of the most rewarding things that you can achieve as an HP41 programmer. Originally reserved only for HP who was the exclusive producer of physical plug-in ROMs, there were later possibilities for companies and user groups (the PPC ROM!) to create a hardware based module. When details of the internal operations and low level code (we call it machine code or mcode, the assembly language of the HP41) became available the first hardware boxes emulating ROMs (first with EPROMS, later with QRAM as emulated ROM) were produced and the possibility of user created mcode started to become realistic. But it does not have to be mcode all the time. A ROM can also be filled with user code programs, like we started to do with putting our DICE program in HEPRAM. It can be very useful to put your own collection of user code in such a ROM so you do not have to rely on magnetic cards or other media.

In this section we will not go into the details of mcode programming itself, but rather focus on the process of getting things done (in the HP41CL of course) and finally adding your own ROM to the IMDB, the internal database of modules, ready to be used by the PLUG function.

We will now go through a number of possible ways to create your own ROM image. The ultimate result of all the options is that we have a ROM image that we need to include in the IMDB and to maintain that in case of possible updates. We will handle that later.

First choice that you have to make is if you really want to do all the work of developing your own ROM on your HP41 calculator itself by using tools that are only available on the calculator. This is very well possible and that is how I did that in the early 80's myself, as I did not even have a PC then. Today we have a number of other options and you should consider to do the hard work on a PC (Windows, Linux or Apple can be used), but it depends on the type of ROM you want to create. In short, here are your options:

- Do everything on the HP41CL: this works great for user code programs in combination with HEPAX and HEPRAM, or when making small modification to an existing ROM

- Use V41, an HP41 emulator running on a Windows PC: does anything your HP41 (C, CV or CX) does and is a great way to test mcode as it has mcode tracing. It does not emulate HP41CL specifics (like YFNX). Recommended for testing mcode and great for making user code ROMs with emulation of HEPAX and HEPRAM.
- SDK41 is an MS-DOS based assembler and linker with a text based mcode source code. You must install an MS-DOS environment (FreeDOS or DOSBox for example), and combining mcode with user code is not so easy
- Calypsi (used to be NutStudio, NUT is the nickname for the HP41 processor), available for Linux (Debian and Arch), MacOS and Windows. Supports mixing mcode and User Code (Focal) and includes a debugger. Today this is my personal tool of preference for creating an mcode ROM

When testing newly developed mcode there is a serious possibility of locking up your calculator. Unless you are using HP41CL dependent hard- or software I would strongly advise that you use a PC-based emulator for the first tests just in case you need to remove batteries to recover from a locked HP41CL. And keep in mind that there are really good emulators for mobile devices (for Android and i-Phones), although transferring a ROM image between devices may be tricky. My setup of choice is Calypsi with the V41 emulator. When the first large mistakes in the code are fixed I then move the ROM image to the real hardware (which could be my MLDL2000, Clonix or NoVRAM or the HP41CL) for further testing.

In case you do not have your own HP41CL, your best option for running your own ROM is to use Clonix or NoVRAM. The MLDL2000 (which I designed myself) is not available for sale anymore.

Once you have your ROM image tested on an emulator (as much as possible) the next step is to transfer the ROM to the HP41CL. We will deal with that in a later section.

13.1. A ROM with user code only

When you want to put your own user code in a ROM image then things are pretty easy. The recommended way to create your own ROM is the to use HEPAX in combination with HEPRAM. I would advise in that case to install only one HEPRAM page as it will be easier to isolate that from the HEPRAM chain and make it fully standalone. Another advantage is that you will know when the module space is filled up. When using two HEPAX modules additional user programs will automatically be placed in the second HEPRAM.

In the next part I will not list step-by-step instructions as you should now have the base knowledge to make this a success.

The source of your programs could come from a Card Reader (watch the port, it goes into Page E, the lower Page of Port 4), mass storage (take care because your IL module uses Pages 6 and 7), barcodes from a book or PPC Journal or just keying in.

You may want to give your own ROM a proper name that appears in the catalog functions. This should be done before all other steps. How to do this is described in Vol 2 of the HEPAX manual starting on page 153. Please take care that the example also takes out the Page from the HEPRAM chain so you cannot use **HSAVEP**. But **HEPINI** (in ***PWRX***) will take it back into the chain again. The function name is the first in the FAT, so this step must be taken first before you add any other user programs. This Volume 2 of the HEPAX manual is a must read for everyone starting to create your own ROM image!

To add your programs to HEPRAM simply use **HSAVEP** as often as you need, in many cases this will work well. Just be careful if one of your programs calls a local (named) subroutine or another program that will be in the same

ROM image, as you may take advantage here of the XROM number. Calls referring to an XROM number are much faster as the system does not have search all ROM's for that one name. The problem here is that **HSAVEP** destroys the XROM information and translates the XROM to an XEQ with a text label, so this will be slower. In that case use the function **XQ>XR** (Execute to XROM) in ***OSX3***. This will convert the XEQ back to XROM again, but the program has to be in HEPRAM already! This requires some planning as you need to load the programs used as subroutine first, or complete your ROM and do the **XQ>XR** on all programs. Needless to say, you must have decided on the final XROM number of this ROM before starting this whole process! Please check by looking at your user code (in HEPRAM!) to verify that the XEQ's are properly converted, otherwise run **XQ>XR** again as it may need an extra iteration in case of a local named label.

When your user code ROM is finished and tested (remember to test with your user code programs removed from main user memory) it is time for the next step, and that is to isolate the ROM from the HEPRAM chain, even if it was only one Page. When plugging your ROM later in a configuration with HEPAX present your ROM may be detected as HEPRAM when you did not 'unchain' it before, and this will confuse the HEPAX setup. And we have a little tool for this in the POWERCL EXTREME module (***PWRX***). So plug this in your 41CL and execute the function **RLSRAM** (Release HEPAX RAM) with the Page number. You now have your own created ROM image in HP41CL SRAM, and it is plugged. To save it in FLASH and add it to the ***IMDB*** go to section 15.

13.2. A ROM developed on the HP41CL

This is the old school method, and practically the only possible way for HP41 hobbyists in the 80's. HP had professional tools available for customers wishing to create their own ROM's, but these were not available to the general user community. There is a ROM named Asssembler 3 (and its successor Assembler 4), but I have not found any documentation for this. It is in the ***IMDB***, so feel free to give it a try, and please share your experiences. There is also the famous ZENROM which had a disassembler to translate existing code in ROM or QRAM in readable mnemonics. But to enter mcode one still had to type in hexadecimal codes which had to be manually taken from a table of instructions, helped by some functions to calculate jump distances. The ZENROM manual is great reading stuff for mcode programmers.

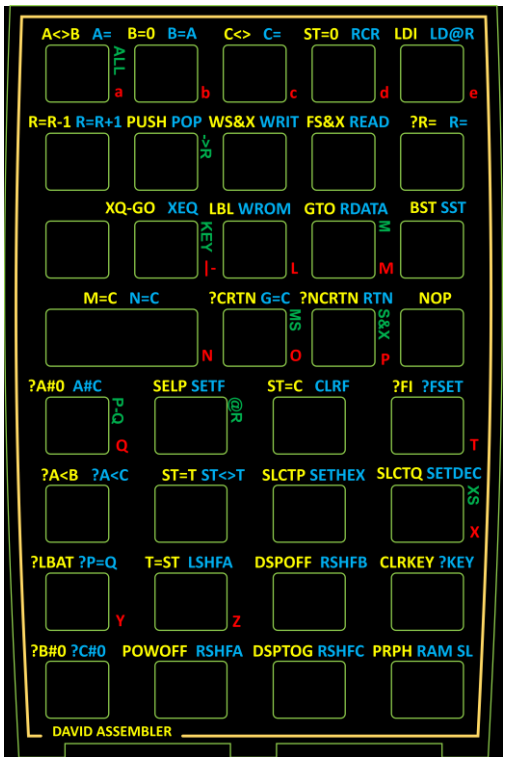

FIGURE 5 DAVID ASSEMBLER OVERLAY

In my view the summit of mcoding tools came with the DAVID Assembler (it has Dutch origins, like myself). It came with a companion ROM containing labels for the HP41 Operating System, supported local labels and allowed typing in mnemonics instead of hex codes and was very well documented.

For developing a ROM on your own calculator DAVID Assembler is definitely the way to go, but keep in mind that you are likely to lock up your HP41 when experimenting with mcode on your calculator.

Together with the ***OSX3*** functions **DISSM** (Disassembler) and **HEXED** (the HEX editor we have used earlier) we are nearly prepared to start writing our own ROM from scratch, but there are some other functions available to make your life easier. Together with Frits Ferwerda, another HP enthusiast from our local HP41 group, (he sadly passed away in 2020) we created our own ROM with supporting functions for mcode development. My own version was ML ROM (and it is in ***IMDB*** with the ID ***MLRM***), which contains a number of tools to facilitate mcode

programming in combination with the DAVID Assembler. There are some more advanced tools, like the RAMPAGE Toolbox with overlaps with -ML ROM, but some functions that are quite handy are not in the RAMPAGE Toolbox. Just be careful to plug **MLRM** in a higher Page than **OSX3**.

Creating your own ROM from scratch requires a bit of preparation:

- In my current configuration I will have my real HP-IL module (with printer enabled) physically plugged in my HP41CL. Main reason is that I use a PILbox (developed by Jean-Francois Garnier from France) to connect to my PC over USB. On my PC I run his program ILPer that emulates mass storage and a virtual printer. The listings (also from CAT"0) below are actually copy and pasted from the printer emulator!
- Collect the tools and plug these in your calculator. My favorite setup (actual list from the pyILPer printer emulator)

```
CAT"0
PGCAT
3:-EXT FCN 3B
4:-LIB#4-R58b.
5:-TIME  3B
6:-PRINTER 2E
7:-MASS ST 1H
8:-OSX BANK2
9:-ML ROM
A: NO ROM
B: NO ROM
C:DAVID-ASSEM
D:MNFR-LBLS
E: Q-RAM          (how to get this is shown in step 2 below)
F:-YFNX 4C
```

Getting an empty page, which is going to be your own ROM to be written from scratch, requires a bit of preparation. First of all, an empty page in FLASH means that is filled with all `0xFF` values, all bits are set to '1', and this is unprogrammed FLASH. Programming this type of memory is only possible by changing a '1' to '0'. It is not possible to program the other way around, this requires erasing FLASH, which is done per sector which spans multiple pages.

The HP41 requires a block of QRAM to have all '0' values, especially in some of the critical areas, and this must be in SRAM as well. A Page with only zeroes is an HP41 empty Page.

In our previous experiments with HEPAX we have used blocks of SRAM in page `0x808` and `0x809`, and those contents are still there! So let us now use page `0x80A` for our virgin ROM Page. We need to make certain that this SRAM block contains al zero's *before* plugging it in a Page.

Step 1: enter in ALPHA the string **"80A000-0000"** and **XEQ "YMCLR"**
The function **YMCLR** (in ***YFNX***) clears a 4K block (starting on a 4K boundary!), its actually fills the block with the last 4 digits in ALPHA.

Step 2: **XEQ "PLUG" "-80A" "E"**
This now plugs the (empty) page `0x80A` in HP41 Page E. Now do a **CAT"0 G**, and the listing will now show Q-RAM for Page E, exactly what we wanted.

I will not go into the details of creating your own ROM, but I will give you a bit of a flying start in the following steps. Please follow these exactly.

Step 3: **XEQ "ASSM"** to start DAVID Assembler
It will remember the location you last visited, if you do not see a prompt **"CONT. _ _ _ _"** for 4 digits use [backspace] to get that prompt. ***DAVA*** uses a hexadecimal keyboard now, so no need to go to ALPHA mode. Enter **E000**, the first location of our own ROM.
If you are in USER mode you will see **NOP**, this is the disassembled `0x000` which is at this location. In USER mode you have a fully reassigned keyboard that we will not use now, so switch off USER mode and you will see **E000 000 @**
The first word in a ROM is the XROM number, so think about this for a moment as you need to decide what this is going to be. You may need to get used to ***DAVA***. When typing in a word, it actually enters that word in the next memory location. So you now have to **[BST]** to move to `0xDFFF`, then enter **010** (or any other suitable number for your XROM) and you will then see
E000 010 P
You can achieve the same with **HEXED** (from ***PWRX***) and then you can enter the number at the prompt when the target address is shown.

The first function in a ROM is the name, so we will take care of that first. Also this requires a bit of planning, as the FAT (Function Addres Table) needs to be at the start of the ROM. This table has a maximum of 64 entries (of 2 words) and we need three extra words: one for the number of functions and two words to end the FAT. This means that, assuming all function entries in the FAT will be needed, the first real free address is at E084. If you anticipate less functions in your FAT then you may of course put the function name elsewhere.

Step 4: in DAVA, move the address to **E084**, and enter the following hex codes (or use **HEXED** if you prefer that):

```
E085  08D   "M"
E086  00F   "O"
E087  012   "R"
E088  020   " "
E089  019   "Y"
E08A  00D   "M"
E08B  02D   "-"
E08C  000   NOP      start of function, instruction visible in USER mode
E08D  3E0   RTN      instruction, visible in USER mode
```

The last two are instructions in HP41 machine code. If you use ***DAVA***, switch to USER mode and you will see the disassembled instructions. We have simply added a NOP and a RTN. When a NOP is the first instruction in a function it means that the function is not programmable. The ROM name can be a real function doing things in case you want it do something.
Your ROM now has a name, but it does not show yet when you do a **CAT"0 G**. The reason is that the function still needs to be added to the FAT.
Functions names, or sub-headers in a FAT start with a "-" (dash or minus sign) by convention. The HP41CX catalog function only lists ROM names with a minimum length of 8 characters. All function names are 'upside down' and the last character (in the lowest memory location) has `0x80` added to the character code to indicate the end of the name. The HEPAX Manual Vol2 explains this in much more detail.

Step 5: in ALPHA enter **"00E08C"** and **XEQ "CODE"** then **XEQ "AFAT"**
This is a handy function that adds the function at the address `0xE08C` (the address of the first instruction) to the next available entry in the FAT (and it will increase the number of functions in the module, the second word in the ROM). Do not worry about the leading 0's, refer to the MLROM description for details of the function. The **CODE** function sort of translates ALPHA into a hex code in the X-register (this is a Non-Normalized Number or NNN) that **AFAT** needs. **Now do a CAT"0 G**

You have now created a real ROM image with one function. It is up to you to explore the art of creating HP41 machine code. Must reads for this are:

- HEPAX manual Vol 2
- ZENROM manual
- Ken Emery's HP41 Machine Code Made Easy

Sooner or later you will need to have your ROM saved on another computer and/or added to FLASH and the ***IMDB*** for easy plugging and unplugging. This step will be described soon.

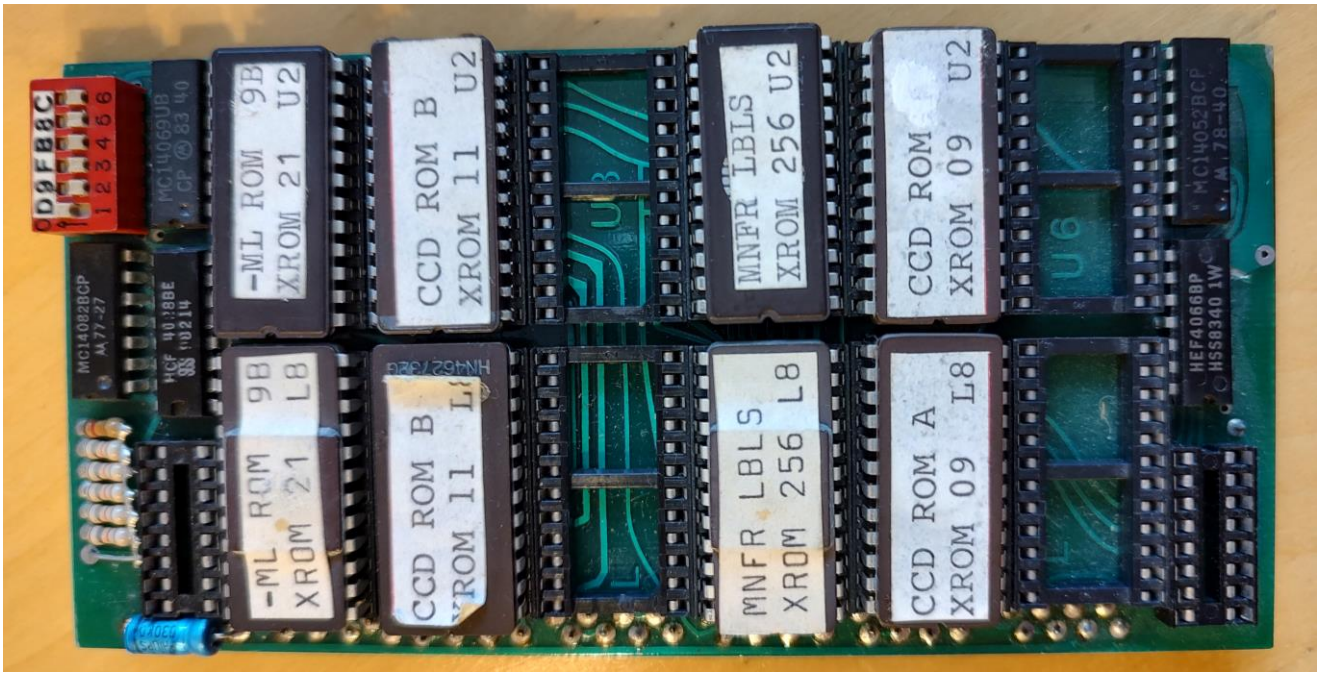

FIGURE 6 EPROM BOARD OF AN OLD SCHOOL MLDL

13.3. A ROM with mcode developed on your computer

Developing a ROM on your host computer has many advantages and really is my preferred way. Quick changes or patches on an existing ROM are best done on the 41CL itself, but even then I would start by getting an emulator installed on the host computer. Which emulator depends on the host computer type you have. There are some for Apple, Linux and Windows. Since I only have a Windows PC I will focus on that.

Advantages of using a host PC for ROM development:

- No risk of losing contents on MEMORY LOST, empty battery or your programming mistakes

- Documented source code (if you care to write comments) in your favorite editor even with syntax highlighting
- Several tools to work with: SDK41 (needs DOS) or Calypsi (available for Linux and Apple as well)
- Test code on an emulator (V41 or EMU41), both have mcode debugging and mcode single stepping. My personal preference here is V41 and both support a virtual HP-IL link that allows connection to your real HP41 if you have an real HP-IL interface and a PIL-BOX. The PIL BOX is a smart little device that connects a PC with USB to an HP-IL loop.

I will not go into any further detail of the development process when using your PC, but rather focus on the process of transferring developed ROM images from the PC to the HP41CL (and the other way around), adding to the IMDB and programming your ROM in FLASH.

There is one special trick that I want to share here. You can **PLUG** a ROM image twice in different ports. When it is a QRAM module this is very useful for developing bank switched ROMs, by plugging one instance into the target bank and page, and a second instance in an accessible page where you can edit with **HEXED** or David Assembler. **HEXED** does support multiple banks, but ***DAVA*** does not, and that makes it very easy to patch code. Editing in one page is immediately reflected in the banked Page!

14. Getting information in and out your 41CL

One of the great features of the HP41 ecosystem was always the possibility to get things in and out of the system. Primarily with the plug-in modules, but do not forget the printer and the HP-IL system. I also was a happy user of the Wand barcode reader. Fortunately the HP41CL offers I/O with a serial RS-232 interface, and that is a must for users. It helps to move ROM images from a host PC to the 41CL, to save the same images and to allow many other ways of data exchange. One of the most important uses is to update the ROM Images in the HP41CL. It does require a serial interface on your host computer and today this will typically be a USB to serial adapter which are cheap and easy to get. In my own HP41CL I have actually built this adapter inside, and my CL has its own mini-USB port ready to connect to my computer.

FIGURE 7 HP41CL WITH HP-IL AND PILBOX

Another spectacular way of connecting the CL to a host computer is by using the HP-IL module. You will need a PILBox which connects the IL-loop to USB (actually a USB simulated serial port). Software is available for Windows, Linux and Apple that emulates HP-IL devices on the computer such as a printer, mass storage or display. These virtual devices can be extended over a network (or in the same computer) to connect with many other emulated devices. The big advantage of using HP-IL is that this is fully integrated in the HP41 ecosystem and that is works the same on your 41CL, a 'normal' HP41 and on many emulators. Any old software written for this will still work. On the PC end it is not too complicated to extract a stored ROM image (or user program) from an emulated mass storage device. The disadvantage is that two Pages are used by the HP-IL module, and even if you do not use the printer, a disabled printer now blocks Page 4 where you might want your Library 4 instead.

The advantage of using the serial interface on your 41CL is that communication is very direct. You simply send or receive a block of memory by indicating start and end address after starting the proper software on the host computer. And this is exactly where the challenge lies, as the available host software is typically limited to sending or receiving data or ROM images. There is a great tool to do exactly this especially for updating your 41CL but I have found no easy way to transfer user programs, Extended Memory files or simulate mass storage using the 41CL serial port. As a result I use both HP-IL and the serial interface. In this section I will focus mainly on using the serial port.

The software on the host computer is a bit critical here. The HP41CL website refers to a number of programs that can run on a Windows PC, Linux or Apple. These are written in C, Java, python, Ruby or available as an executable (for Windows that is). My personal favorite is the clupdate program (written in Java with multi-platform support) written by Silvain Cote. This works only for transferring complete ROM images and is an absolutely great tool to update your 41CL in case there are new or changed ROM images.

14.1. Transfer of User Programs

Now this is a real challenge, and so far I have not found an easy solution with the HP41CL only. If you have an HP-IL module and PILBox than things are much easier, as you can connect a virtual drive and share that between the 41CL and an emulated HP41 (V41 or EMU41). Or you can use the emulated printer to simply list the program and copy the result in an editor. The best multi-platform software here is called pyILper (Python version of ILPer, and ILPer is the interface software for the PILBox), a link is in the references. This allows full emulation of an HP-IL drive, and on the host side you can click and select the individual files on the drive and save or export on the host file system.

If you do not have the setup with an HP-IL loop then generating a program listing, or a binary version of the user code (this will typically be a .RAW file, the default standard file format for a HP41 User Code program as used by the V41 emulator) is pretty tricky, apart from finding the exact memory location of the program and exporting that over the serial port. This would still require some editing on the host side. I do need to mention the USB41 module (designed by Clonix and NoVRAM creator Diego Diaz), a hardware HP41 module with USB interface that can emulate a printer on the host computer that allows you to create listings of programs.

Very recently I have published ***YPRT***, a modified PRINTER ROM, that prints over the serial port to a connected computer, using Diego's printer emulator (under Windows) on the host computer. You can simply print a program listing to the emulator, and copy the listing to an editor and store it.

In my view the best option without additional hardware is to create a HEPRAM module with your user code program(s) and transfer that complete module image to the host PC (this process is described in the next paragraph) and then import in an emulator (V41 for example) for printing and documenting, or exporting as a RAW file. Many steps unfortunately.

14.2. Transfer of complete ROM images

FIGURE 8 MY 41CL WITH INTEGRATED USB TO SERIAL PORT

The HP41CL is very well equipped to send and receive complete ROM images or blocks of 41CL memory, and several solutions for the host computer exist for various computer types. Just as easy is sending blocks of data from 41CL FLASH or SRAM by giving the function the correct addresses. Receiving data from a host PC is a bit different, as data can be received *only* in SRAM, and this requires some planning as you need to know ahead where this data or ROM image can go to.

On a Windows PC I typically use CLreader and CLwriter as the command line is short and easy to use for simple tasks. On the 41CL you can best use **YIMP** en **YEXP** (both in ***YFNX***), these receive and send blocks of 41CL memory up to 4K words using the serial port. Before you can use the serial connection and transmit data you need to make some preparations:

- Connect the HP41CL to your host PC (typically with a USB to serial adapter) and find out which serial port is used, you need to know this when starting the program on the PC. In my case this is COM3
- Decide on the baudrate. 4800 baud works reliable, 9600 baud is known to work when sending data from the 41CL, it is usually not reliable when the 41CL receives data. Best is to settle on 4800 baud all the time. There is no handshake protocol available and your good old calculator needs enough time to process data. In some cases I have to use 2400 baud when sending data from a PC.
- Initialize the serial port on the HP41CL
- Start the program on the host PC when receiving data on the PC
- Start the transfer on the 41CL before the host program times out when sending data. This means that you normally need to prepare the 41CL (address in ALPHA) even before starting the PC program. If you want to receive data on the 41CL then start the transfer on the calculator first, and then on the PC.

Step 1: **XEQ "SERINI"**
XEQ "BAUD" 48
Assuming that you have the cable connected between your 41CL and the host computer you have now enabled the serial port and set the baudrate to 4800. We will send our own ROM named "-MY ROM" that we prepared earlier, so first we need to prepare the parameters in ALPHA
Enter in ALPHA **"80A000-0FFF"**
This is later used for **YEXP**: the start address is our page `0x80A` that was the designed QRAM (the full start address is needed) and `0xFFF` is the transfer length minus 1. So we will transfer a full 4K Page.

Step 2: **XEQ "YEXP** but do NOT complete the command!

At this point another detail of the HP41 memory structure is important. The physical memory in the HP41CL is 16 bits wide, and the HP41 ROM (or QRAM) memory width is 10 bits, and we will transfer 4096 words of 2 bytes each. On the other side (at least when using CLReader) the result will be a so-called .ROM file of 8 Kbytes long that can be read by an emulator. Each 10-bit HP41 ROM word is stored in a 16 bit word in this file. Another complication is the byte order (endian) of the host computer, but this is usually taken care of by the CLReader program (in this case at least).

Many emulators (and also the DM41X hardware, which is a great recreation of the HP41) do not handle .ROM files, but want a .MOD file instead. Tools are available to convert between these formats, which is not trivial unfortunately (see references). To confuse things even more, there is a somewhat older .bin format which is a packed .rom file. My own ROMHandler program (Windows only) can read, analyze and convert between these file formats (see references).

Step 3: on the Host computer: **CLReader myrom.rom COM3 4800**
Replace COM3 by the port relevant for your situation
This will typically be in a cmd shell (on Windows) and you will have about 25 seconds to complete the command on the HP41CL, so complete the command and you will see **SENDING** in the display

At the end of the transfer (after about 20 seconds for a 4K ROM image) you can have a quick view in the file at the receiving end (with a hex editor for example) to verify that it indeed contains the bytes you have typed earlier in "- MY ROM". The functions **DLD48** and **UPL48** in the ***PWRX*** module automates part of the up- and download process.

Receiving a ROM image or block of data is almost the same, just make certain that the memory location is in HP41CL SRAM and that that memory block is not used by something that should not be overwritten. It is possible to write to a page that is plugged in. Sending data may be from a FLASH page but you cannot write to a page in FLASH!

14.3. Transfer of a hardware ROM image

A bit of a special case is to import (or export!) a ROM image from an existing physical module or a NoVRAM/Clonix or (legacy) MLDL or EPROM device and this process requires some special preparations depending on the following use cases

- Regular port addressed hardware module, like MATH. The Page of the module is defined by the port it is plugged in. If you plug a single page module in Port 3 for example (in my HP41CL Port 1 is taken by the serial interface and I have a TIME module in Port 2) than it will be in Page C. In case of an 8K module (PPC ROM for example) these are Port addressed and take both Pages of a Port.
- Some hardware modules are hardwired to a Page, like the Printer, Time and HP-IL modules. These modules are already available in the IMDB of course, but there could be surprises
- Multi-bank modules (like the HP Advantage module) are port addressed and have one or more banks that are 'shadowed' behind the primary bank and not immediately visible
- Clonix/NoVRAM, MLDL2000 and legacy RAM or EPROM boxes may contain multiple ROM images that are more or less hardwired to a port. If you can influence the configuration you have full control, but that is not always the case and some experimenting is needed

The more info you have available for a ROM type the better. Otherwise create a minimal HP41CL configuration with ***OSX3*** (and ***4LIB*** of course), as this give you the best tools to get more information, and the first function to use is of course **CAT"0 G** to find out what the exact Page configuration is. Put ***OSX3*** in the lowest Page possible, otherwise any active entry points in your physical ROM could mess up the ***OSX3*** operation. After that plug ***PWRX*** in a free page and you are ready to go, but keep in mind that ***PWRX*** also depends on ***YFNX***! Some systems can contain many ROM images with limited space and you have to be creative then. Remember that with the MMU disabled you still have ***YFNZ*** available. After you have done some investigation and have noted the Pages with interesting information you can use **YMCPY** to copy from a physical port to a physical location in SRAM (this SRAM page does not have to be plugged of course).

With the functions **BANKED?** And **BANKS?** (in PWRX) you can check if there are pages using multiple banks. Now copying one of the banked pages is a bit of a challenge, or at least it used to be. There is a little gem hidden in Ángel Martins CL Expanded Memory Functions (ID is ***XPMM***) module that allows you to copy any bank from one page into another page, this is **CPYBNK**. (there is also a **CPYBNK** in ***YFNF***, the 41CL Memory Functions module). In this case you must have a valid QRAM page plugged in your 41CL and you may have to tweak a bit with available pages.

Writing a ROM image to your hardware device in the first place requires that the device has QRAM. This could be a NoVRAM, MLDL2000, W&W RamBox or even a real HEPAX module (lucky you if you have one!). This use case can be a bit of a challenge as in some cases the program doing the actual writing to QRAM must be located in the device itself! This is certainly the case for a NoVRAM device. It will work on an MLDL2000, and I do not have access to the other hardware types to be able to test this. Important here again is the proper mapping of the QRAM Page to be visible from the 41CL, and in most cases you can use **YMCPY** to copy a complete page to the target QRAM.

15. The almost final chapter: the IMDB and FLASH memory

The last task ahead of us now is to get some basic understanding of the IMDB and how to add your own ROM image to it and finally to put it in FLASH with no risk of losing it after a power loss.

The IMDB (Image DataBase) is much more than a simple list of ROM Images, it contains metadata per ROM and allows grouping in case of multi-page or multi-bank ROMs. Its companion is the FLDB or 41CL FLASH Database. The FLDB is mainly used in the automatic update tool and there should be no reason to manipulate it in the scope of getting your own ROM sorted out in FLASH.

The IMDB contains a lot of information and is described in detail in the 41CL manual. The most important items that we have used before are the ROM image name and page in FLASH. The name is always 4 characters and of those characters the first and last are unique identifiers. The two middle characters can be used freely to create an easy to remember name. The chapter Image Database in the 41CL manual has all the details and most important for us is a table with the first and last characters of the name and if these are in use or not. Some numbers can be used as well. In order to choose a name pick a free spot in this table and fill in the middle characters by choice. In general you should not use "9" as the first character as this is reserved for other purposes, we will touch this subject briefly. Looking at the table my choice for a name would now be QRAM, as our own rom in its current state could serve as a template for any new ROM image in QRAM, ready for further editing. So we now need to prepare our own ROM for more permanent storage, and we need to do a few more things to the ROM contents to make it complete. So get ready to do a bit more editing with either David Assembler or **HEXED**.

Earlier we have made one entry in the FAT of our ROM image. At the end of the ROM there are a few entries that could be useful, this is the ROM Revision and the ROM Checksum. The ROM Revision is only used for

documentation and is an easy way of checking the version, the checksum is not very critical, it is just nice to keep that up to date but the system will not complain if it is not correct.

Step 1: start **HEXED** or **ASSM** and go to address `0xEFFB`
Enter the following hexcodes and be very careful not to enter anything in address `0xEFFA`!

0xEFFB	**031**	**"1"**
0xEFFC	**030**	**"0"**
0xEFFD	**019**	**"Y"**
0xEFFE	**00D**	**"M"**

I prefer **ASSM** (David Assembler) because it shows the characters as you type. You have now typed in the revision code MY-01. If and how you maintain this is not critical, but is a nice way of keeping track of the version of your ROM.
Go out of **ASSM** (or **HEXED**) and run the **CHKROM** program (from ***OSX3***), at the prompt type 16 (the XROM number) and you will see the ROM revision that you just typed in. Display will show **16 MY-01 BAD**. You can see the same when running **"XCAT"** form ***MLRM***.

Step 2: enter **"E"** in ALPHA, **XEQ "CODE"** and **XEQ "ROMSUM"** (these are all from ***MLRM***)
The last command is from MLRM, and requires to have the Page number coded in the X-register. Run **CHKROM** again to verify if the ROM Checksum is now correct.
The checksum in the ROM is not very critical and many ROM authors do not care to keep this up to date, so it is really up to you to decide how to handle this

You now have a nice ROM Image in QRAM located in Page E in your HP41, and in 41CL Memory page 0x80A. You could send this to your PC for archiving if you like, or do a final test on an emulator, but basically this is now a ready to go ROM image.

0x3DC	ALL_FF			empty	0x53D36BD2	
0x3DD	ALL_FF			empty	0x53D36BD2	
0x3DE	ALL_FF			empty	0x53D36BD2	
0x3DF	ALL_FF			empty	0x53D36BD2	
0x3E0	ALL_FF			Reserved for User	0x53D36BD2	
0x3E1	ALL_FF				0x53D36BD2	
0x3E2	ALL_FF				0x53D36BD2	
0x3E3	ALL_FF				0x53D36BD2	
0x3E4	ALL_FF				0x53D36BD2	
0x3E5	ALL_FF				0x53D36BD2	
0x3E6	ALL_FF				0x53D36BD2	
0x3E7	ALL_FF				0x53D36BD2	

FIGURE 9 SNAPSHOT FROM MEM_REF

Now we need to get back to some IMDB basics. Our ***YFNX*** offers a number of functions to search the IMDB and to work on an IMDB copy in SRAM. This is really useful for adding entries to IMDB before committing to FLASH. Keep in mind that the **PLUG** and **PPLUG** functions only work with data from the IMDB in FLASH, it will not use the data from the copy in SRAM! So we cannot test plugging in this way. Our goal is to have an entry in the IMDB in FLASH, and to have our own ROM image in FLASH as well. Take another look at the mem_ref table and you can see that the User reserved space (for a v5 HP41CL) starts at FLASH page `0x3E0`. Many other pages are marked as unused. We mentioned earlier that empty FLASH are all 1's as this is easy to program. Erasing FLASH actually means setting all bits to 1 and requires a special process. If we are going to overwrite the IMDB itself, the entry for our own ROM must also contain all 1's! An the IMDB has its own place in FLASH at page `0x0DF`.

Our first big step is to program our own ROM, which we will call QRAM, into flash in the first empty FLASH page. Our source is in RAM at `0x80A`, and the destination is in FLASH at `0x3EO`.

Step 3: **"80A>3E0"** in ALPHA and take a deep breath
Now **XEQ "YFWR"**
And that was it already, you have copied a 4K block to FLASH. To confirm, do the following:
Enter **"3E0000-0000**" in ALPHA, and **XEQ "YPEEK"**. Look in ALPHA and you will see our XROM number: **"3E0000-0010"** !
In case you do not believe it, do an **YPEEK** at address `0x3E1000` (the next page), and this will return `0xFFFF`, this is unprogrammed FLASH memory

The next steps are not really necessary to add the new ROM to the IMDB, but show some of the nice features. First we will make a copy of the IMDB in SRAM in a designated location for this, page `0x805` or the extra functions buffer and practice with some of the functions.

Step 4: **XEQ "IMDBCPY"** to copy the IMDB from FLASH to SRAM
XEQ "IMDBR", this marks the RAM copy of IMDB as active the next operations (but not for **PLUG**!) will apply to the copy in SRAM. When you are not certain if the copy in SRAM or the original in FLASH is active you can test by **XEQ "IMDBF?"** which will reply with YES if the FLASH copy was active.
The function **IMDB?** queries the selected IMDB for a match with the ROM image ID entered at the prompt.
XEQ "IMDB?" "QRAM" will result in a display like **"Q**M-F-FFF"** as this image does not (yet) exist in the database. Try **XEQ "IMDB?" "MATH"** and the result will be **"MATH-0-034"**. The 034 here refers to the page in FLASH where the ROM image is located, 0 is the image type, in this case a 4K single page image.
Now try another trick: **XEQ "IMDB?" "Q???"** you will now get a list of all IMDB entries starting with Q and everything after that in alphabetical order. You can stop the list with **[R/S]** and then **[SST]** or **[BST]** to browse through the list and **[backspace]** to exit. When you exit the list the last ID is left in ALPHA and you can use this with the **PPLUG** function with the Page number in X.
The ROM Images in the IMDB are categorized (see the list of categories in the CL Extreme manual) and with **XEQ "IMDB?" "?MAT"** you will get a list of all mathematics related ROMs.
Please do read the section Image Database Functions in the CL Extreme manual as there are some more useful tricks!

Step 5: in ALPHA type: **"QRAM-0-80A"** which is the argument for the next function
And then **XEQ "IMDBINS"** and **XEQ "IMDB?" "QRAM"** : **"QRAM-0-80A"**
This adds (and verifies) our QRAM entry with a page address of 0x80A to the IMDB (in SRAM!). Now we need to ask ourselves if this is what we really want. In case we PLUG a module like this, it will always point to an image in RAM. This could be meaningful if you always want to use this image in RAM, but it must be there also after a power loss to ensure a successful plug. In any case we have already programmed our ROM in FLASH as an example of a finished ROM so let's proceed that way.

Step 6: in ALPHA type: **"QRAM-0-3E0"** where 0x3E points to the page in FLASH where our ROM is, type is 0
And then **XEQ "IMDBINS"** and **XEQ "IMDB?" "QRAM"** : **"QRAM-0-3E0"**
We now have the correct entry in our IMDB, but this is still active in SRAM. A **PLUG** operation will not work.

Now things get really interesting, and we have a number of items to keep in mind. Remember that an unprogrammed entry in the IMDB consists of four words with all values `0xFFFF`.

- The **IMBDINS** function will also work when the FLASH version of IMDB is active. So we can actually re-activate the IMDB in FLASH again (with the IMDBF function) and do **IMDBINS** function, but there is no room for

mistakes here. Once programmed, the bits in FLASH can only be written from '1' to '0' and correction from '0' to '1' is not (easily) possible. It is OK to overwrite an existing value in FLASH with the same value, the net result will be no change.

- The function **IMDBUPD** will write the contents of the SRAM copy back to the IMDB FLASH version, and your changes will be implemented. Make no mistakes here!
- There is no function to remove an entry from IMDB, you can do an **IMDBINS** with page number 000, this will generate a NULL entry. To completely remove an entry you need to manually set all 4 words in the correct memory location in the IMDB to `0xFFFF` (in SRAM). To change this in FLASH follow the process described at the end of this section
- **IMDBINS** works fine for simple configurations as you can only set the image type. In our example this is 0 for a single 4K image, and this will be sufficient in most use cases. Refer to the CL Extreme manual for other module types. To set other parameters for your ROM image, such as the application type (mathematical, astronomy, etc.) or page restrictions these need to be manually edited in the IMDB entry. All this is pretty well explained in the HP41CL manual, chapter Image Database.
- The IMDB entries are addressed by a combination of the first and last character of the 4-character ROM identifier. The middle two characters are stored in the entry itself. This is the reason why searching an image by ID is very fast, and searching by page is much slower. Again, study the HP41CL Manual for all the details.
- Our SRAM copy of the IMDB sits in a designated buffer, that may be used by other functions. RAM page `0x805` is used for the IMDB copy

We are nearly ready to commit our IMDB in FLASH, but be aware that the QRAM entry will be removed when an update is done of the IMDB image itself (with the auto updater process) and it is necessary to make a note of the parameters used to create this IMDB entry. Unfortunately I have not found any other way to preserve this information apart from having your ROM added to the official repository by contacting Monte.

Now we will update the FLASH based IMDB, and this can be done with the **IMDBUPD** command or the **IMDBINS** command. The latter is preferred as this is much faster than writing a full 4K image which IMDB is. The advantage of **IMBDUPD** is that you can test and change it in SRAM before committing to FLASH.

Step 7: in ALPHA type: **"QRAM-0-3E0"** where 0x3E points to the page in FLASH where our ROM is, type is 0
XEQ "IMDBF" to make the IMDB in FLASH active again.
XEQ "IMDBINS" "QRAM" to add your own QRAM to the IMDB and verify this with **IMDBF?** (to check if indeed the FLASH copy of IMDB was active) and **XEQ "IMDB?" "QRAM"**

You have really done it, you have created your own ROM image, programmed it in FLASH and added to the Image Database. Now for some proof the last steps in this section. We will do that by unplugging the RAM based version of the QRAM image and plugging the newly made FLASH version

Step 8: **"XEQ "PLUG" "" E** where the double quotes mean to press **[ALPHA] [ALPHA]**
This unplugs the ROM from Page E (where our ROM image still is).
XEQ "PLUG" "QRAM" E to plug the FLASH version in the same Page E. Now start DAVID Assembler in our ROM with **XEQ ASSM**. It will probably take you immediately to the memory location that was used the last time with DAVA. If this is indeed somewhere in Page E try changing the value. Cool isn't it? The **NO WRITE** message means that writing is not possible, in this case because the image in in FLASH.

This concludes this section of flashing your own ROM image and registering it in the IMDB. Almost. One important topic is how to update something which is already in FLASH memory. A second topic that we will briefly touch is the CRC checksum and the (automatic) update process.

When you need to change something which is in the HP41CL FLASH memory there are some critical steps that you need to follow. I have mentioned this before, but let me repeat it to be complete., FLASH memory has some important characteristics:

- When writing to FLASH you can only chance a bit value of 1 to 0
- FLASH memory is erased with a special type of write sequence
- Erasing FLASH means that all bits are set to one, all words will be `0xFFF`
- FLASH memory is erased by sector, and the sector size in the HP41CL is 32K words, or 8 ROM images. Erasing a sector takes about 6 seconds.
- Writing to FLASH and erasing FLASH takes some more power

The consequence of the FLASH characteristics is that if you need to correct a value in FLASH memory it is necessary to first copy the complete sector where that value is to SRAM, correct the value (in SRAM), erase the sector and them write the entire sector back to FLASH. The good news here is that the 41CL memory map has a designated part of SRAM reserved for the copy of the FLASH sector, 8 pages in from `0x810` to `0x817`. When you do need to go through these process I would advise you to use functions in the CL-Update ROM ***YUPS***, these save you from many manual keystrokes and possible errors.

In our example we have the QRAM image all alone in a sector in FLASH, in page `0x3E0`. This is an uncomplicated case as you can simply erase the sector and then program a new version of QRAM in the same position. Obviously you must make certain that you have a copy somewhere in a safe place and that you have prepared and tested the updated version.

WARNING: The next example is a critical operation and may invalidate your 41CL configuration with the result of an update being necessary. I have tested it and it works, but there is no room for mistakes. When you are uncomfortable with this skip this example until you really need it.

ERASING and programming FLASH memory consumes more power and will not work with the BAT indicator on. Some rechargeable battery solutions lose power without activating the low battery indicator and this must not happen during an erase or write operation to the FLASH memory. Ensure that you have enough battery capacity or use external power when doing these exercises.

In this example we will correct a mistake that I once made. When playing with the IMDB copy in SRAM I removed an entry, effectively setting the page address to `0x000`. I then did an **IMDBUPD**, which overwrites the copy in FLASH, and sets the same entry in FLASH to `0x000`. This was not what I wanted to do and I now had an invalid IMDB in FLASH that needed to be replaced. If you ever find yourself in such a situation you can follow the steps below. First of all you must prepare a correct version of the IMDB by copying the version from FLASH to SRAM with **IMDBCPY** as described earlier in this section. Test it and then test it again. The new IMDB is now in SRAM page `0x805`.

Another complication to keep in mind is that any ROM that is in the FLASH sector that will be erased must NOT be plugged. In case this ROM is absolutely needed during the process it must be copied to a suitable location in SRAM, and the SRAM version must be plugged. A check of the other ROM images in the same sector is a vital step here. The FLASH Sector Erase function **P8ERASE** (from ***YUPS***) will actually check for this condition. During the

erase cycle no FLASH access may be done, and **P8ERASE** and the **YFERASE** function in CL Extreme (from version 2A) automatically take care of this.

Step 1: **XEQ "PLUG" "YUPS" D** to plug the CL Update ROM in Page D (assuming that is a free page).

Step 2: Enter in ALPHA **"0D8"**, this is the first page of the FLASH sector that contains IMDB (which is at 0DF). **XEQ "P8BFR"** this copies 8 pages from `0x08D-0x08F` to the SRAM sector buffer `0x810-0x817` The **P8BFR** function is one of the useful tools in ***YUPS***, otherwise we would have to copy 8 pages with individual commands.

Now comes the tricky part. The updated IMDB must be copied to the correct location of the FLASH sector buffer. In case you are updating anything else than the IMDB double check where in the sector buffer the image is that needs to be replaced. In the case of the IMDB this image is copied from page FLASH page `0x08F` to `0x817` in SRAM, and that is where we need to have our updated IMDB.

Step 3: Enter in ALPHA **"805>817"** and **XEQ "YMCPY"** to copy the new IMBD to the correct location in sector buffer. When in doubt about the result of the copy operation you could use **YEDIT** (in PWRL, Power CL Utilities) to look at a number of locations and check XROM numbers or other familiar content.

WARNING: in the next step a complete sector in FLASH containing the IMDB will be erased and reprogrammed. Without a valid IMDB you will not be able to plug ROMs by name and direct programming of the MMU will be needed to recover from this situation.

Step 4: Enter in ALPHA **"0D8"** and **XEQ "P8UPD"**, and press **[R/S]** to confirm. This function does the complete update using the data in the sector buffer including the sector erase. Progress is shown in the display and finished with a short TONE. Needless to say, double check the target sector address before proceeding.

Step 5: **XEQ "IMDBF"** to activate the IMDB copy in FLASH and **XEQ "IMDB?" "QRAM"** to verify if the IMDB entry is now correct. If that is the case then cleanup your configuration by unplugging ***YUPS*** and any other tools that you do not immediately need.

A topic related to the manipulation of IMDB is the ROM CRC checksum and ***FLDB***, the 41CL Flash YCRC Database. The 41CL YCRC checksum of a ROM image differs from the checksum at the end of a ROM image. This 41CL CRC is much more sophisticated and can be calculated using the **YCRC** function in ***YFNX***. The FLDB contains the YCRC of all images listed in the IMDB and is only used in the automatic updating process. In my humble opinion it is not necessary to keep a record of your own ROM image CRC's unless you want this included in the automatic updating process. I have not found any tools to add your own YCRC to ***FLDB***. The best option to fully support the automatic updater is to have your ROM added to the official repository.

On the topic of the automatic updater, I try not to make this too complicated. I let the automatic updater (***YUPS*** with the PC program from Sylvain Cote) run for the full 2 to 3 hours. After that I manually update my own personal ROM images using **IMDBINS** and that's it. The update process is very well described and I will not treat this further here.

16. A few HP41CL specific goodies

The HP41CL gives us huge amounts of memory at our disposal, and this can be used in several smart ways, and the tools for that exist already. In this section we will focus on two specific HP41CL features: managing different configurations and Expanded Memory.

16.1. Expanded Memory in the HP41CL

In the HP41 world we know Extended Memory, offered to us by the Extended Functions module and its companion Extended Memory modules. This is already included in your HP41CL and uses the full available HP41 memory map (see Table 3 - HP41 User Memory) and maps to the first part of the 41CL SRAM in page `0x800`. Three more pages (`0x801` to `0x803`) are reserved as Alternate Memory (as this is called in the ***YFNF*** 41CL Memory Functions Manual) or Expanded Memory, and this is ideal to make a complete or partial copy of the contents of User Memory. A number of functions will allow you to exchange information between your own User Memory and the Alternate Memory sections and this enables you for example to replace the contents of all Extended Memory with just one command, like plugging a completely new set of these modules. This is highly recommended reading stuff starting at page 18 of the 41CL Memory Functions Manual. Please take care that ***YFNF*** has an XROM conflict with our own QRAM module image, so you may want to unplug ***QRAM***.

The 41CL Expanded Memory Utilities ROM (the ID is ***XPMM***) takes this concept even a step further by providing individual register access to data the Alternate Memory area.

You could of course also program a block of registers into FLASH memory, but the exact addresses must be manually defined, but saving the entire User Memory to FLASH (or any location in SRAM) is really very simple. Remember that your User Memory sits in a 4K page in SRAM at `0x800`, and the following shows how simple it really is:

Step 1: Enter in ALPHA **"800>3E1"** and **XEQ "YFWR"**,
Of course you must ensure that the FLASH target page at `0x3E1` is empty (all words `0xFFFF`), but that is really all you need to do. I have chose the FLASH page just behind our ***QRAM*** image. *ALL* your User Memory is now saved, including all your FOCAL programs, Extended Memory, Key Assigments and the Status Registers (stack, ALPHA, flags etc). This is the ultimate backup, although the ***XPMM*** image has more sophisticated functions that will do a validity check of some backups.

Now do some changes, set flags, delete a program and change the size.

Step 2: Enter in ALPHA **"3E1>800"** and **XEQ "YMCPY"**, you now have the HP41 User configuration back that you had a few minutes ago. And really, this includes all of the User Memory, including your flags, stack, Extended Memory and Key Assignments!

With the Alternate Memory you can change HP41 User configurations on the fly. By saving the same in FLASH the configuration can be recalled, but any changes cannot be written back again to FLASH without a FLASH erase cycle.

16.2. Keeping track of your 41CL contents

To know what is exactly in your HP41CL is quite a challenge. We have seen the possibility to list the IMDB contents and that list can be very long. The ***YFNF*** ROM has a few functions that will help you find out if a page in FLASH or SRAM is used or not, but it will not tell you what is in it. Use the functions **MEMCHK** or **PMEMCHK** to

find out quickly if a page is used at all, and then hope to find its use quickly on one of your sticky notes. I think there is need for a good CAT-like function that can actually give a hint of the contents of a page.

Good helper functions are **BFREE** and **BUSED** in ***PWRX***. These list available or used Pages.

16.3. Managing 41CL ROM configurations

We have used some tools to store and retrieve MMU configurations in chapter 11, and learned there (I hope) that the 41CL can save up to 15 secondary MMU registers, and that the primary MMU registers is always the active set. A basic set of functions is in ***YFNX***, which I recommend should be in your HP41 almost always. The earlier mentioned ***YFNF***, or 41CL Memory Functions, extend this concept by offering named configurations that can be auto-initialized with a number of pre-defined ROM combinations for engineering, math, programming, games and more. This can be useful as is, or a start to create your own set of configurations. Use the function **CFGINI** to initialize these MMU sets. These configurations can be changed with the **PLUG** command from ***YFNX*** and the appropriate names starting with 9. ***9ELE*** will load the ROM configuration for electrical engineering, which is MMU set 9. Details are described in the 41CL Memory Functions Manual from page 32. The first three available MMU sets would then be ***9CFA***, ***9CFB*** and ***9BAS*** (BAS for Basic), for many users this will be enough. Of course you can still use the other ***9xxx*** names for your own configuration. ***YFNF*** does not need to be plugged once the configurations are initialized, only ***YFNX*** is needed to plug and unplug ROM images. Testing your configuration is a must as there is always a possible XROM conflict.

To backup your configurations remember that all secondary MMU's (and many more 41CL specific settings) are in SRAM page `0x804` and we have seen earlier that it is not very complicated to save a copy of all this in FLASH or elsewhere in SRAM.

When mixing configurations with physical modules you must keep in mind that the presence of a real module (when plugged in Page 6 or higher) is detected by ***YFNX*** and this page is then locked and the MMU entry disabled to allow access to the module. The lock status is sticky, and the Page remains locked when the module is removed and these Pages must be manually unlocked, otherwise you will most likely get a **LOCK ERROR**.

Also ***YFNX*** locks itself, but allows itself to be moved to another Page when loading a new configuration (if that is specified in the configuration!).

17. Afterthoughts

More than 40 years after the introduction of the HP41 there are still new developments, thanks to a number of enthusiasts who still produce new ROM images and share tips and tricks and take newer users by the hand. I am glad to be part of that community and also somewhat proud of my contributions. The HP41CL and also the DM41X have given a new boost to activities, and we have to thank the creators of those additions to the HP41 universe. I hope that this document will make it much easier to master the HP41. Feedback is much appreciated.

Meindert Kuipers, Spring 2022.

FIGURE 10 MY COUNTRY OF BIRTH, COUNTRY OF TULIPS

18. References and credits

Many thanks to Monte Dalrymple for creating the HP41CL and its tools and documentation, and for proofreading this document. Ángel Martin created many ROMS that are great for using not only on the HP41CL, but on all other variations of the HP41, and he also did the proofreading.

Below is the list of important links.

Reference	URL	Description
41CL Homepage	http://systemyde.com/hp41/index.html	Home page for the 41CL Calculator
41CL Manuals	http://systemyde.com/hp41/manuals.html	Original HP41CL manual, including mem_ref. The 41CL Calculator Manual is also the reference for YFNZ
41CL Other Docs	http://systemyde.com/hp41/documents.html	Includes the OSX3 manual
41CL File area	http://systemyde.com/hp41/files.html	Download links of the current ROM repository and various tools, which includes the CL Updater and clreader/clwriter plus some other goodies
HP Museum Forum	https://www.hpmuseum.org/forum/index.php	The best place to get help and general information
HP41.org	www.hp41.org	The best source for manuals, books, ROM images
V41 emulator	HP41.org and https://hp.giesselink.com/v41.htm	HP41 emulator for Windows
PIL Box	http://www.jeffcalc.hp41.eu/hpil/index.html	HP-IL to PC USB based link
HP-IL emulation	https://hp.giesselink.com/hpil.htm	Emulated hardware on your PC, to use with the PIL Box or the V41 emulator
HP-IL emulation	https://github.com/bug400/pyilper/blob/master/INSTALL.md	pyILPer, a python based HP-IL emulator with printer and mass storage emulation
EMU41	http://www.jeffcalc.hp41.eu/hpil/index.html	HP41 emulator, DOS based
Clonix, NoVRAM and HP82143 emulator	https://www.clonix41.org/	Hardware plug-in configurable modules, and a printer emulator
Calypsi	https://www.calypsi.cc/	Modern mcode and RPN toolchain with debugger for Linux, MacOS and Windows
SDK41	www.hp41.org	Old-school mcode toolchain for DOS
ROMHandler	https://github.com/mjakuipers	Repository for my own HP41 creations, including this document

Modules and documentation used in this document. All modules mentioned are in the HP41CL module repository

Reference	Documentation	Documentation available at
YFNX module	Manual for the 41CL Extreme Functions	41CL Manuals
MATH, ***PPCM***, ***41AD*** modules	Manuals for the MATH, PPC ROM and Advantage modules	HP41.org
OSX3 and ***4LIB*** modules	AMC-OSX and Library4 modules with documentation, by Ángel Martin	41CL Other Docs
HEP2 and ***HEPR*** modules	Modules and Manuals for HEPAX. HEPAX manual Vol 2 is especially recommended for (starting with) mcode programming. When using on the HP41CL, use the patched version HEPAX-4H (or higher)	HP41.org
PWRX module	POWERCL Extreme, by Ángel Martin, Used for HEPRAM initialization	41CL Manuals
YFNF module	41CL Memory Functions	41CL Manuals
ZENR module	ZENROM, many Mcode tools running in a module. The manual is great reading for mcode programmers	HP41.org
ASMB module	Assembler 3, various Mcode tools	HP41.org
DAVA module	My favourite for creating mcode on the HP41, comes with a companion ROM contain all global label in the HP41 OS	HP41.org
MLRM module	ML ROM, my own ROM with tools for creating mcode and ROM's. Developed together with Frits Ferwerda	HP41.org
YPRT module	HP41CL serial printer module	41CL Other Docs
XPMM module	CL Expanded Memory Functions	41CL Other Docs
YUPS module	41CL Update functions, automatic updater to keep your 41CL ROM collection up to date	41CL Manuals
Book	HP41 Machine Code Made Easy by Ken Emery, really good introduction in HP41 mcode programming	HP41.org
Book	Inside the HP41C, by Jean-Daniel Dodin, a wonderful detailed description of the entire HP41 system, both hardware, mcode and user code structure	HP41.org
Book	HP41CX, A Programmers Handbook, by Poul Kaarup	HP41.org