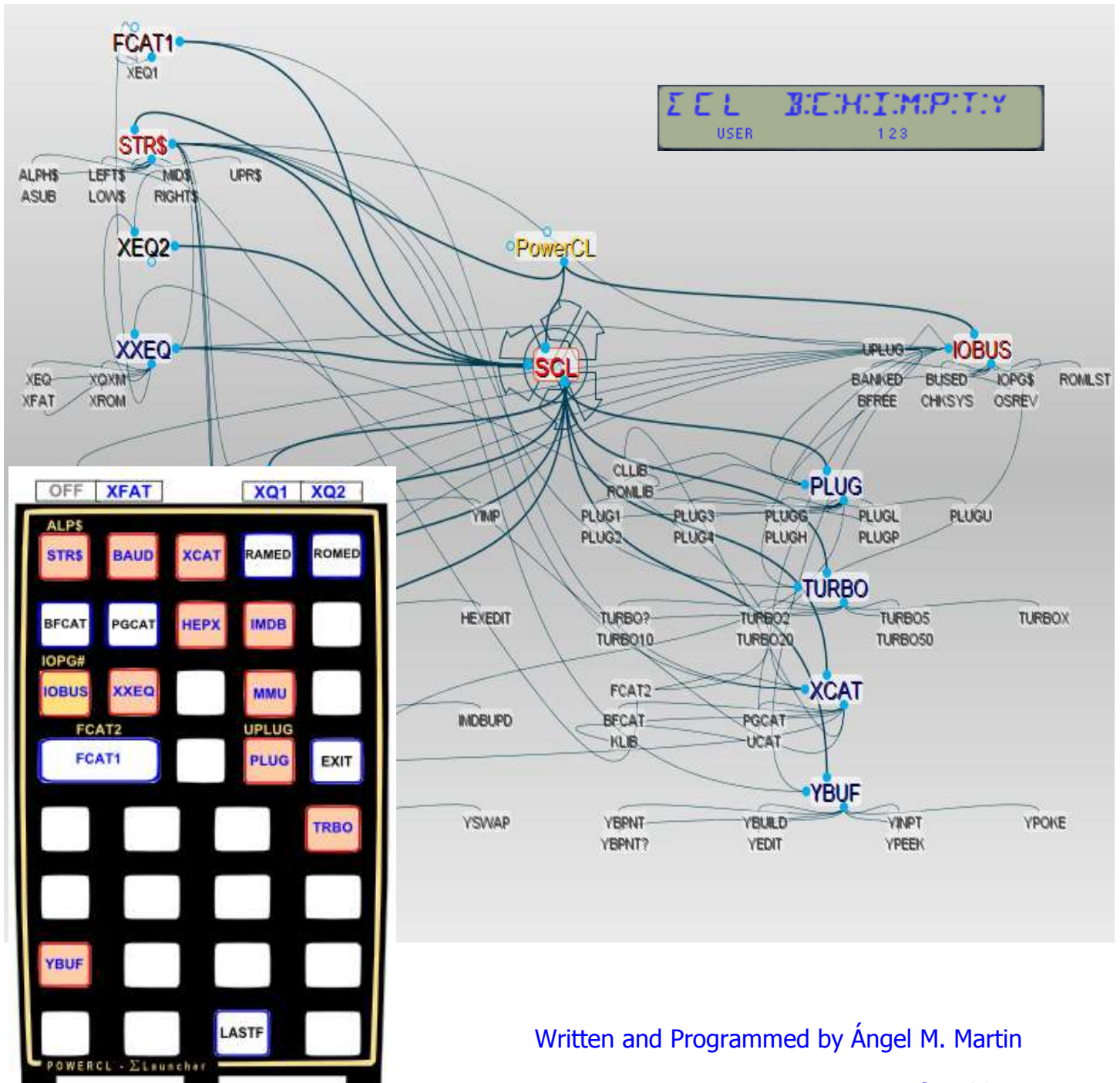


POWERCL_EXT Module

System Extensions for the 41CL Revision – 02

User's Manual and QRG.



Written and Programmed by Ángel M. Martin

November 2014

This compilation revision 4.W.10.9

Copyright © 2012 -2014 Ángel Martín

Published under the GNU software licence agreement.

Original authors retain all copyrights, and should be mentioned in writing by any part utilizing this material. No commercial usage of any kind is allowed.

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow.
See www.hp41.org

CLWRITE Source Code written by Raymond Wiker.

Cover and inside graphic from Personal Brain 7.1, see <http://www.thebrain.com/>
Inside photos © Geoff Quickfall; Jürgen Keller, 2011

Acknowledgments.- This manual and the POWERCL module would obviously not exist without the 41CL. Many thanks to Monte Dalrymple for creating the amazing CL board.

Everlasting thanks to the original developers of the HEPAX and CCD Modules – real landmark and seminal references for the serious MCODER and the 41 system overall. With their products they pushed the design limits beyond the conventionally accepted, making many other contributions pale by comparison.

POWERCL_EXT Module

Table of Contents.

1. Introduction

1.1.	Preamble: Fatter than FAT!	5
1.2.	Introduction.	6
1.3.	Page#4 Library and Bank-Switching	6
1.4.	The PowerCL Overlay	7
1.5.	Plugging the POWER_CL module	8
1.6.	The Functions at a glance	9

2. The functions in detail

2.1	Function Launchers	15
2.2	Plugging and Unplugging	17
2.3	The "Last Function"	19
2.4	Extended XEQ	21
2.5	Sub-function Groups and FATs	23
2.6	Catalogues and CATALOGs	25
2.7	Buffer Catalog	26
2.8	Interrogating the MMU	28
2.9	Page Catalog	29
2.10	A wealth of a Library	30

3. HEPAX and Security

3.1.	Configuring the HEPAX system	33
3.2.	HEPAX chain alteration	36
3.3.	Page-Plug Functions	37
3.4.	Using Module Info as input	39
3.5.	Security Functions	41
3.6.	Encrypting RAM contents	42

4. Advanced Territory

4.1.	Alternate memory blocks: Expanded memory	43
4.2.	Y-Input and Y-Reset	46
4.3.	Image Database functions	47
4.4.	Using Page #4	48
4.5.	Editing RAM areas with RAMED	49
4.6.	Editing ROM areas with ROMED	50
4.7.	Quick & Dirty sRAM Editor	52
4.8.	Moving and Swapping MMU entries	53
4.9.	Calculator Flash backup	54
4.10.	Downloading ROM images	57

5. I/O Bus and Page#

- 5.1 [The system as a whole](#) 59
- 5.2 [The pages within](#) 61

6. Strings and Alpha Extensions

- 6.1 [String Manipulation functions](#) 63
- 6.2. [Alpha and Display Utilities](#) 64

7. System Extensions

- 7.1. [X-Memory Utilities](#) 69
- 7.2. [Second Sets of Main and X-Memory](#) 71
- 7.3. [Buffer Handling Utilities](#) 74
- 7.4. [Key Assignment Utilities](#) 76
- 7.5. [Page Functions revisited](#) 77
- 7.6. [XRAY: how big is your lollypop?](#) 78
- 7.7. [Flag handling functions](#) 81
- 7.8. [Other Miscellaneous Utilities](#) 82
- 7.9. [Extra Functions – Con't.](#) 84

8. Unit Management System

- 8.1. [Constants Library](#) 87
- 8.2. [Unit Conversion Catalog](#) 90
- 8.3. [Unit Conversion Comparison](#) 93
- 8.4. [Farewell.](#) 96

9. Appendixes.

- 9.1. [Summary of \$\Sigma\$ CL functionality](#) 98
- 9.2. [Detailed CL ROM id# table](#) 100
- 9.3. [FOCAL Program Listing](#) 105
- 9.4. [MCODE Highlights](#) 106
- 9.5. [Serial Transfers CLWRITE / CLREAD](#) 109
- 9.6. Checking ROM Configurations 114
- 9.7. Re-allocating MMU Entries 116
- 9.8. Breaking the FAT barrier 117
- 9.9. Dr. Jekyll and Mr. Hyde 120



POWERCL_EXT Module Extension Functions for the 41CL

0. Preambles: Fatter that FAT!

This manual covers revision "O2" of the PowerCL_Extreme module, an extended implementation with a full four-bank configuration on just a 4k footprint. There are 242 functions and numerous advanced capabilities packed in this module, ranging from strictly CL-related to practically every other aspect of the 41 system. This manual should provide utilization instructions for the functions, but not an exhaustive treatment of the subjects involved. A thorough documentation of every single one would easily take hundreds of pages and require much more in-depth treatment of the topics.

To access all this many functions they have been structured into sub-function groups, each of them with an auxiliary FAT. These "hidden" FATs are cleverly interconnected by launchers located in the main page, real gateways into the bank-switched pages – true parallel dimensions of the module.

All functions are programmable using a non-merged approach, and thus can be called individually – using their index within the group as a second line parameter for the function launcher. All this happens *automagically* when typing the sub-function name, and its complexity is totally transparent to the user – a beauty to behold.

The programmers of the HEPAX module (real MCODE grand-masters) developed this impressive technique to overcome the 64-functions limit in the FAT. The same implementation has been used here, adding further capabilities to the functionality that build upon the original design - making it easier to use and a little more powerful still.

Broadly speaking this is the breakdown of their contents:

- a. The contents of the Main FAT (also known as **FAT-0**) are essentially the same as in previous versions of the module, covering the CL-specific functions and some of the most relevant system extensions – notably the *Unit Management System* plus a *Constants Library*. Prompting functions are also located here. In addition to that, there are three launcher functions to access the other sub-function groups, both by name and by index within their FAT. There are a total of 64 functions in FAT-0.
- b. The first sub-function group (also known as **FAT-1**) includes new functions to manage the CL Expanded memory – that is three sets of alternate blocks in RAM to make back-up copies of your Main and Extended Memory. Also included in this page is a set of extra functions, many taken from the ToolBox module - general utilities and mixed topics that will be covered later in the manual. There are a total of 84 functions in FAT-1.
- c. The second sub-function group (also known as **FAT-2**) includes many of the functions from the Rampage and Alpha modules. By functional area, there are all the buffer management, Extended Memory enhancements, and of course the Alpha functions group. Also the HEPAXA function group from the HEPAX module is included – in recognition of the original masterpiece. There are a total of 86 functions in FAT-2.

It comes without saying that the real challenge lies not in stuffing all that functionality into the banks, but for it to be useful there must be capable navigation techniques and usability features. Here's where the function launchers (**XXEQ**, **XQ1**, **XQ2**, and **XFAT**) as well as the new **FCAT** sub-function catalogues fulfilled the expectation; in their role of master of ceremonies and organizers of the module - more about this later on.

1. Introduction.

Without a doubt the 41CL can be considered in many accounts to be the pinnacle of the HP-41 system. It comes with a well thought-out function set to manage its capabilities, from the basic to the more adventurous – which has inspired the writing of yet further extensions to that capable toolset.

This module is designed to enhance and complement the *YFNS Extreme*, a.k.a. **YFNX** function set, providing easier access to the many powerful capabilities of the 41CL platform. Some are function launchers, grouping several functions by their area of functionality into a single, prompt-driven one – like it's the case for the Plug/Unplug functions, the **BAUD** rate, **TURBO** and **MMU** settings functions. A "launcher of launchers" sits atop these, providing quick access to multiple YFNX and other functions *from a single key assignment*.

Some others extend the functionality by providing new features and more convenient alternative to manual tasks. Examples of these are:

- A fully-featured ROM library catalog system, allowing direct plugging into the port of choice
- The *Page* plug functions (alternative to the *Port* ones), including routines to handle page #4.
- HEPAX configuration and set-up, making the HEPAX integration a simple and reliable affair.
- Security functions to password-protect your machine from prying eyes.
- Two powerful RAM and ROM editors, a hacker's real delight.
- Comprehensive Expanded Memory enhancements of alternate RAM blocs.
- An extended implementation of the Unit Management System, with Electrical units support and featuring an all-new Constants Library.
- Tons of sub-functions in auxiliary FAT groups... and many more!

Other housekeeping functions roundup the set, making for a total of 242 functions tightly packed into a bank-switched 16k ROM. This was a design criterion, as the small footprint of the module (just one page, or effectively 4k) makes it ideal to combine with other utility packs, most notoriously the CCD OS/X (or its alter-ego AMC OS/X) for the ultimate control - and except some honorable exceptions there is no duplication between the two.

Page#4 Library and Bank-Switching.

The first thing to say about the PWRCL_EXT module is that - like the POWER_CL and previous CL_UTILS revisions - it extensively uses routines and functions from the Page#4 Library. Make sure the Library#4 revision "M" (or higher) is installed on your system or things can go south. Refer to the Page#4 Library documentation to properly configure the Library#4 before you start using it.

Plugging the Library#4 on the CL is done using **YPOKE** to directly write its image location in the MMU register for page#4. Assuming the library image is loaded at location 0x83F in sRAM, the syntax is:

enter "**804040-883F**" in ALPHA, then execute **YPOKE**.

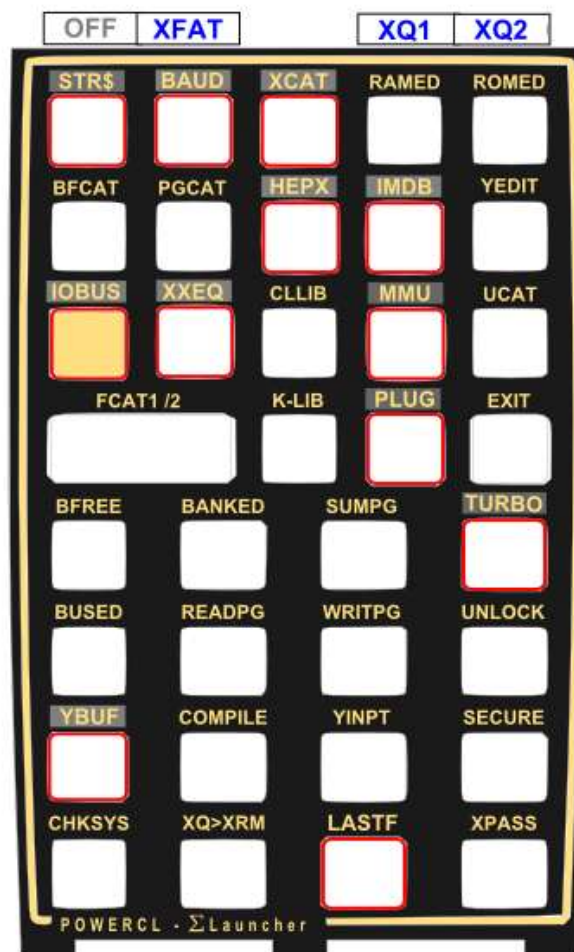
This module is, to the author's knowledge, one of the very few ones using bank switching. The idea of using an auxiliary bank originated from the usage of the very long ROM image internal table required by **CLLIB** and **ROMLIB**. Even after completing the page#4-aware version of CL_UTILS, it was clear that a bank-switching version would be the ultimate solution, which also enabled an enhanced implementation of the Unit Management System (UMS) to be included.

Subsequent versions doubled the number of banks (expanding from 2 to 4), and added the sub-functions launchers, FAT catalogs, and many other functions to the list – touching a large variety of functional areas: Buffer management, Extended Memory enhancements, Alpha functions, HEPAX functions. And last but definitely not least the CL Expanded Memory functions for backup.

The main launcher Σ CL was tweaked to also connect with the FAT groups via the dedicated sub-function catalogs and launchers: **XQ1**, **XQ2**, **XXEQ** and **XFAT** – enough to make your head spin but yet easy to grasp. *We can say this is the all-wheel drive version, 4x4 – with the four banks using the Library#4.* Bank switching is managed behind the scenes by the launchers. Each function is responsible for returning to Bank-1 upon completion. The user needs not to be concerned with all this power and simply use it seamlessly.

The PowerCL Overlay.

To facilitate using the different functions their launchers have been arranged taking the first letter of the name as the designated key to invoke and execute them. This is easier to remember than other approaches; even if those hot keys are not grouped together in contiguous locations (like say, the two top keys). An overlay is therefore just an additional convenience more than a necessity - but here it is, in case you fancy those over the prompting cues.- The overlay shows all functions available at the Σ CL prompt – with only one key assignment required.



- Function names on dark grey background (keys with red circle around them) represent the main launchers. Each of them will present further options in its prompt, as will be described later in the manual.
- Function names on standard black background are available as individual options due to their frequent need (for a power user). Some of them are also included in the launcher prompts for additional convenience.

Plugging the POWER_CL module.

Being a bank-switched module, there are four 4k-blocks involved: all four banks need to be loaded using the appropriate syntax at the **PLUG** function prompt. Whether you use a copy in sRAM or it's already burned in FLASH, make sure that the four banks occupy contiguous blocks, or the **PLUG** command will not work properly.

For example, assume the banks are copied into locations 0x840 to 0x843 on the V3/V4 CL board (any other group of contiguous addresses will also work, even in V2 systems). Then this will be the syntax required to plug it in page #9:

Execute: **PLUG, "*840", pg#**

You can use page#6 or page#7 on V3/V4 CL boards, (obviously without a Printer or the HP-IL connected) as *the module is designed to be completely compatible with those system locations*.

Using the new Image Database functions.

Version V4 of the CL board comes with the POWERCL_EXT module already burned in flash, thus can be plugged in simply by using its mnemonic "PWRX" at the **PLUG** function prompt. Note that this will NOT take care of the Library#4 dependency, which you'll have to do manually as well.

To create a new entry in the IMDB for POWERCL_EXT you can use function **IMDBINS**, once the four banks are loaded in sRAM. You'll need to provide an address and mnemonic, making sure you specify the "-16k" type. Refer to the CL manual for instructions and the proper syntax.

WARNING: Be aware of the dependency with the Library#4, which requires it to also be configured for the proper operation of the functions. Failure to do this will cause surely unexpected and likely unpleasant results. Refer to the Library#4 documentation for details.

WARNING: PWRCL_EXT is designed to work paired with YFNX. This is a mandatory requirement.

A word of caution.

As wise men remind us, "*with power comes responsibility*". Indiscriminate usage of some of these functions can have unpleasant consequences, ranging from unexpected results and easy-to-recover machine lock-ups to more serious ones, losing HEPAX data. Functions have some built-in protection to ensure that they're used properly, but they are not absolutely foolproof in that such protection can always be circumvented. So beware, and as general rule "*if you don't understand something, simply don't use it*".

To assist you with this, the more dangerous functions are marked with **WARNING** signs all throughout this manual. Avoid them if you're not absolutely sure that you know what they are for, and fully understand their operation. Note that the flash back-up functions are back in. They were available in the CL_UTILS module but were later removed in previous versions of the PowerCL. Notice that the *hard-coded addresses used now are incompatible with V1/V2 revisions of the CL board* - They may be too risky for casual users; so be forewarned! And always follow the "better safe than sorry" rule.

It had to be said – so now that we got it out of the way we're ready to dive into the POWERCL description and usage example. May you have a nice ride!

Function index at a glance.

Without further ado, here are all POWERCL functions, in the three FATs.

#	Function	Description	Inputs	Output
0	-PWRLC_EXT	Splash Screen	none	Shows message
1	XCUI _	Main CL Global Launcher	Prompts "B:C:H:I:M:P:T:U"	Launches selected Launcher
2	ADRID	ROM id# from address	Flash addr in Alpha	ROM ID in Alpha
3	BAUD _	Baud functions launcher	Prompts "1:2:4:9:I:M:X"	Launches selected function
4	CLLIB _	ROM Library Alphabetical	Prompts "A-Z"	Starts listing at selected letter
5	DLD48	Download at 4800 baud	Destination address	ROM downloaded from PC
6	HEPINI _"	HEPAX FileSys Init	Prompts for values	Initializes HEPAX File System
7	HEPAX _	HEPAX Launcher	Prompts "4:8:6:I:D"	Launches selected function
8	HPX4 _	HEPAX Config 4k - CL	Prompts for page#	Configures 4k HEPAX on CL
9	HPX8 _	HEPAX Config 8k - CL	Prompts for page range	Configures 8k HEPAX on CL
10	HPX16 _	HEPAX Config 16k - CL	Prompts for page range	Configures 16k HEPAX on CL
11	IMDB _	IMDB Launcher	Prompts "A:C:I:F:R:U:?"	Launches selected function
12	MMU _	MMU functions launcher	Prompts "C:D:E:G:T:?"	Launches selected function
13	MMUCAT	MMU Catalog	None	Sequential list of MMU Entries
14	PLUG# _	PLUG functions launcher	Prompts for location	ROM with ID in ALPHA is plugged
15	PLUGG _	Plug Page	Prompts for page	Plugs ROM in page
16	PLUGG? _	Page Location MMU	Prompts for page	Content of MMU entry for page
17	PPG#4 _	Page#4 Plug	Prompts "F:L:S"	Selected ROM plugged
18	ROMLIB _	ROM Library by type	Prompts "E:F:G:M:U:X"	Sequential list of ROMs by type
19	SECURE	Enables password lock	None	Sets SECURE mode ON
20	TURBO _	TURBO functions launcher	Prompts "X:2:5:1:0:;:?"	Launches selected function
21	UNSECR	Disables password lock	Asks for password	Sets Secure mode OFF
22	UPLG4	Clears MMU entry for page #4	None	MMU entry cleared
23	UPLGG _	UPLUG page	Prompts for page "6-F"	Unplugged if present
24	UPLUG\$	UPLUG by Alpha	ROM Revision in Alpha	Unplugged if present
25	UPLGID _	Unplug by ROM mnemonic	Prompts for ROM mnemonic	Unplugged if present
26	XCAT _	Xtended CATalogs	Prompts: "B:F:H:K:L:M:U"	Launches selected function
27	XPASS	change password	Asks old/new passwords	Password is changed
28	YBKSWP	y-Blocks Exchange	Source in X, Destination in prompt	Exchanges alternate blocks (0-4)
29	YCRYPT	En/Decrypts RAM content	Prompts in program	RAM contents scrambled/restored
30	YEDIT _	CL sRAM Editor	Prompts for start address	reviews & edits word values
31	YINPT	Y Input	Prompts for characters	HEX entry plus control chrs
32	YMEM _	YMemory Launcher	Prompts "S:O:E:D:I:M:X"	Launches selected Launcher
33	YMOVE	Moves MMU Entry	FROM: in Y, TO: in X	Moves the MMU to TO: page
34	YSWAP	Swaps MMU Entries	FROM: in Y, TO: in X	Swaps the MMU between pages
35	YRALL	Y-Read-ALL	None	Reads Calculator/MMU from Flash
36	YWALL	Y-Write-ALL	None	Writes Calculator/MMU to Flash
37	-SYS/EXT	Section Header / DTOA	Text in Display	Writes text to ALPHA
38	ASG _	Multi-byte ASN	Prompts for mnemonic	Assignment is made
39	BFCAT	Buffer Catalog	None	Shows present buffers
40	FDATA _	Function Data	Prompts for FNAME	Fnc. Data in Alpha
41	HEXKB _	HEX Keyboard	Prompts for entry	Codes NNN in X, string in Alpha
42	IOBUS	I/O Bus Launcher	prompts "B:F:U:O:R:?"	Launches selected function
43	UPL48	Upload ROM Image	Destination address	Sends image to PC
44	PGSIG _ _	Gets Signature for page	Prompts for page, 1-14	data string in Alpha
45	PLNG _	Program Length	Prompts for P. Name	Program length in X and Alpha
46	RAMED _	RAM Editor	Address in X	Memory edited as per inputs
47	ROM?	ROM Information	Prompts for XROM id#	Header in Alpha, pg# in X, #FNC In Y

#	Function	Description	Inputs	Output
48	ROMCAT __	ROM id# CAT	Prompts ROM id#, 1-32	Stats CAT-2 at this point
49	ROMED _ _ _ _	ROM Editor	Prompts for address	ROM edited as per inputs
50	SIGPG?	Gets page# of matching sig	Signature in Alpha	Page# in X
51	STR\$	String Manipulation launcher	Prompts "L:M:R;S:U:W"	Launches selected function
52	T>BS __	Base Ten to Base	number in X, base in prompt	Converted number in ALPHA
53	XQ1 _	FAT1 eXEQute	prompts for FNAME	Launches function
54	XQ2 _	FAT2 & FAT1 eXEQute	prompts for FNAME	Launches function
55	XFAT _ / _ _	FAT Function# Launcher	prompts for FAT# , FNC#	Launches function
56	XROM _ _ _ _	ROM function Launcher	prompts for XROM#, FNC#	Launches function
57	XSEQ _	XEQ Launcher	prompts "0:1:2:F:M"	Launches selected function
58	-SI	Direct Unit conversion	value in X, string in Alpha	Converted unit in X
59	A<>RG __	Swap ALPHA and Registers	Register in prompt	Data swapped
60	KLIB _	Constants Library	4 displays of 5x each, SST	Constant value to X, unit to Alpha
61	SI-	Reverse Unit conversion	value in X, string in Alpha	Converted unit in X
62	ST<>RG __	Stack and Alpha swap	Stack and Regs. contents	Contents exchanged
63	UCAT _	Unit Catalog	section in prompt (1-5)	Lists units starting at section#
0	-YFNM NEW	Section Header	n/a	n/a
1	?MMU	Is MMU enabled?	none	Yes/No, skips if False
2	?YFNX	Checks for YFNX version	none	Shows error if not present
3	MM-YBK	Main memory to Block	Destination Block# in X	Main Memory saved in block
4	MMYSWP	Main Memory Exchange	Swap Block# in X	Status Regs. Saved in block
5	ST-YBK	Status Regs to Block	Destination Block# in X	Extended Mem saved in Block
6	STYSWP	Status Regs. Exchange	Swap block# in X	Status Regs. exchanged
7	XM-YBK	Extended Mem. To Block	Destination Block# in X	Extended Mem. saved in block
8	XMYSWP	Ext-Memory Exchange	Swap block# in X	Extended Mem exchanged
9	YBK-MM	Y-Block to Main memory	Source block# in X	Main Memory restored from block
10	YBK-ST	Y-block to Status Regs	Source block# in X	Status Regs. Restored from block
11	YBK-XM	Y-block to Extended Mem.	Course block# in X	Ext-Mem. Restored from block
12	YBSP	ALPHA back Space	String in ALPHA	Deletes rightmost character
13	YCL>	Clears string from ">"	String in ALPHA	Clears from ">" char to the right
14	YFNZ?	Location of YFNS ROM	Alternate YFNS?	Disables MMU if not found
15	YMCOPY	General-purpose cOPY	Control data as NNN in X	Copies data to block 0-4
16	YMEXCH	General-Purpose Exchange	Control Data as NNN in X	Swaps Data between blocks 1-3
17	YRESET	Resets the MMU	None	Disables the MMU
18	YSWP>	Swaps both sides of ">"	string in ALPHA	Alpha swapped around ">"
19	Y1SEC	1 Second delay	None	Pauses for 1 second
20	-PAGE FNS	Section Header	n/a	n/a
21	BANKED	Lists banked pages in I/O Bus	none	String in Alpha
22	BANKS? _	number of banks in page	Prompting - page# in X in PRGM	Number in X
23	BFREE	Lists free pages in I/O Bus	none	String in Alpha
24	BLANK? _	Tests whether page is blank	Prompting - page# in X in PRGM	Yes/No, skips line if false
25	BUSED	Lists used pages in I/O Bus	none	String in Alpha
26	CDE	Code NNN	HexCode in ALPHA	NNN in X
27	CHKSYS	Check xROM Configuration	none	OK/BAD message
28	CLBL	Clear Block	"bbbbeeee" in X as NNN	Clears block content (sRAM only)
29	DCD	Decode NNN	NNN in X	HexCode in Alpha
30	FOG	En/Decrypts Page	adr in X (NNN), Pwd in ALPHA	En/Decrypts pg content
31	OSREV	Shows revision of OS ROMS	none	String in Alpha
32	PG? _	Page information	Prompting - Pg# in X in PRGM	Header in Alpha, XROM,# funct. In X
33	PGCPY __	Copy Page	FROM: in X, TO: in prompt	Copies page from/ to
34	PGROOM _	Counts zero-words within page	Prompting - Pg# in X in PRGM	Number in X, stack lifted
35	PGSUM _	Page Sum	Prompting - Page# in X in PRGM	Sums page and writes value

#	Function	Description	Inputs	Output
36	READPG	Reads page from HPIL disk	page# in X, Fname in Alpha	Page read from disk
37	ROMCHKX	Checks ROM	XROM# in x	Verifies checksum
38	ROMLST	Shows all XROMs plugged	none	String in Alpha
39	SUMPG _	Sums Page	Prompts for pg#	Sums page and writes value
40	WRTPG	Writes page to HPIL disk	Page# in X, Fname in Alpha	Page copied to Disk
41	X=PG?	Pages Comparison	FROM: in X, TO: in prompt	
42	XQ>XR	Changes XEQ into XROM	Program Name in Alpha	XEQ changed to XROM
43	-XTRA FNS	Section Header	n/a	n/a
44	ΣRG?	SREG location finder	none	Location in X
45	AVU41	Alpha view as 41 Instruction	Digit in Alpha	Instruction syntax in ALPHA
46	CFX	Extended CF	flag# in X (0 – 55)	Flag is cleared.
47	COMPILE	Compiles User Program	Prgm Name in Alpha	Program compiled
48	CRTN?	Curtain location finder	none	Location in X
49	CSST	Continuous SST	PC in current program	Lists steps in sequence
50	CVIEW	Continuous View	Text in Alpha	Shows text, non stop
51	DREG?	Number of Data Registers	none	Current SIZE in X
52	FC?S	Flag clear and set test	flag# in X (0 – 55)	Tests flag and sets it
53	FREG?	Free registers Finder	none	# of available REGS in X
54	FS?S	Flag Set and Set test	flag# in X (0 – 55)	Tests flag and sets it
55	FSIZE	Drive File Size	File Name in Alpha	File Size in X
56	GETPC	Gets current PC in x	none	PC value in X as NNN
57	GTEND	Go to .END>	none	Sets PC to the .END.
58	LASTP	Go to Last Program	none	Sets PC to last program in memory
59	PC<>RTN	Exchanges PC and RTN	PC location in X	Data exchanged
60	PCOPY	Programmable COPY	Prompts for PRGM Name	Copies PRGM to RAM
61	PUTPC	Puts PC	PC in for in X as NNN	PC is reset
62	READF	Read Data File	XFILENAME,DRVFileName	file copied to Drive
63	READXM	Reads all XMEM contents	Drive File Name in ALPHA	XMEM read
64	RTN?	Is a RTN pending?	None	Yes/No, Skip if false
65	SFX	Extended SF	flag# in X	Flag is set
66	SPEED	Recalls CPU cycles data	none	CPU cycles/sec in X
67	ST<>Σ	Swaps Stack and Σreg	none	reg contents exchanged
68	TOGF	Toggle Flag	Flag# in X	Flg status toggled
69	TGPRV	Toggle Private Status	Prgm Name in Alpha	PRV status toggled
70	WRTDF	Write Data File	"XMFile,DRFile" in Alpha	File is written to Drive
71	WRTXM	Write all X-Memory	FileName in Alpha	XM Contents written to Drive
72	X<I>Y	Indirect X<>Y	reg# in X and Y	Reg contents exchanged
73	XQ>GO	Deletes one return address	none	RTN ADR deleted
74	ZOUT	Outputs Complex number	Im in Y, Re in X	„Z=Re +/- jIm“ in Alpha
75	-56 BITS	Section Header	n/a	n/a
76	ΣDGT	Mantissa digits sum	Number in X,	Sum of digits in X
77	1CMP	One's complement	number in X	result in X, stack drops
78	2CMP	Two's complement	number in X	result in X, stack drops
79	BRXL	Byte Rotate X Left	number in X, #bytes in Y	result in X, stack drops
80	NOTX	Not X	number in X	result in X, stack drops
81	RXL	Rotate X Left	number in X	result in X, stack drops
82	RXR	Rotate X Right	number in X	result in X, stack drops
83	WSIZE?	Word Size Finder	none	word size in X
84	XANDY	X and Y	numbers in X and Y	result in X, stack drops
85	XORY	X or Y	numbers in X and Y	result in X, stack drops
86	X+Y	X+Y	numbers in X and Y	result in X, stack drops
87	Y-X	Y-X	numbers in X and Y	result in X, stack drops
88	Y/N?	Prompts for Y/N keys	Uses control keys	Skips line if "N"

#	Function	Description	Inputs	Output
89	FCAT1	FAT-1 Catalog	Uses control keys	Enumerates sub-functions
0	-POWERCL 3X	Section Header	n/a	n/a
1	BFLNG	Buffer Length	Buf id# in X	Buffer length in X
2	BFLST	Buffer List	none	shows string with buffers present
3	BFVIEW	View Buffer Content	Buf id# in X	Shows all buffer registers
4	BUFHD	Buffer Header	Buf id# in X	Header reg location in X
5	CLEM	Clear Extended memory	none	Main EM directory deleted
6	CLMM	Clears complete RAM	none	Clears all Main and X-Memory
7	CRBUF	Create Buffer	Buf#,size in X	Buffer created
8	DELBUF	Delete Buffer	Buf id# in X	Buffer Deleted
9	FLCOPY	Copies X-Mem File	FROM, TO names in ALPHA	Both files must already exist
10	FLHD	File Header	FileName in Alpha	Register location in X
11	FLTYPE	Gets File Type	FileName in Alpha	Filetype in X
12	GETBF	Get Buffer	FileName in Alpha	Buffer is read from XMEM
13	GETKA	Get Keys	FileName in Alpha	KA Read from XMEM
14	GETST	Get Status Registers	FileName in Alpha	Status registers restored
15	GETZS	Get Complex Stack	FileName in Alpha	Complex Buffer Restored
16	HEXIN	Hex Input	Prompts for Hex chars	Text in Alpha and Display
17	KACLR	Clear Keys and Buffers	"OK/OKALL" in Alpha	Clears KA and Buffers
18	KALNG	KA Length	none	KA length in X
19	KAPCK	Pack KA registers	none	KA registers packed
20	LKAOFF	Suspends Local Keys	none	Local Keys suspended
21	LKAON	Activates Local Keys	none	Local keys activated
22	MRGKA	Merges Keys	FileName in Alpha	Merges KA from XM File
23	PEEKR	Peek Register	Register# in X in decimal	Register content in X as NNN
24	POKER	Poke Register	Reg# in X, Content in Y	Replaces register content
25	REIDBF	Re-issue Buffer id#	oldid#,newid# in X	buffer id# is changed
26	RENMF	Rename File	"OldName,NewName" in Alpha	File Name is changed
27	RESZBF	Resize Buffer	id#,size in X	Buffer size changed
28	RETPFL	Re-types File	FileName in Alpha, Type# in X	File type is changed
29	RSTCHK	Reset Checksum	FileName in Alpha	Checksum byte is refreshed
30	SAVEBF	Save Buffer	buf id# in X, FileName in Alpha	Buffer is written to XM File
31	SAVEKA	Save Key assignments	FileName in Alpha	KA regs written to XM File
32	SAVEST	Save Status registers	FileName in Alpha	Status registers saved in XM
33	SAVEZS	Save Complex Stack	FileName in Alpha	Complex Buffer saved in XM
34	WORKFL	Appends Workfile Name	none	FileName appended to Alpha
35	XQXM	XEQ Program File	FileName in Alpha	Execution transferred to file
36	-ALPHA/DSP	Section Header	n/a	n/a
37	AIN	ARCL Integar part	x in X	int(X) appended to Alpha
38	ALEFTJ	Alpha Justify Left	Text in Alpha	Justified left
39	ANUMDL	ANUM w/ Deletion	String in Alpha	Number from string to X
40	APPEND _	Appends text to Alpha	Text in prompt	Text appended to Alpha
41	ARCLCHR	ARCL Character	FileName in Alpha	ARCLs chr# at current pointer
42	AREV	Alpha Reverse	Text in Alpha	Reversed text in Alpha
43	ASCII	ASCII value of Char	Text in Alpha	Value in X, leftmost chr#
44	ASUB	Alpha Substitute char	chr# in Y, chr# value in X	char is replaced
45	ASWAP	Alpha Swap around ","	A,B in Alpha	B,A in Alpha
46	CHRSET	Character Set	none	Lists all character set
47	CLA?	Is Alpha Empty?	Text in Alpha	Yex/No, skips line if false
48	DSP	Sets Display digits	# of digits in X	Display settings changed
49	DSP?	Finds Display digits	none	number of decimal digits in X
50	DTOA	Display to ALPHA	Display contents	Text in Alpha

#	Function	Description	Inputs	Output
51	DTST	Display Test	none	Shows display all lit up
52	LADEL	Left Alpha Delete	Text in Alpha	Leftmost char deleted
53	LEFT\$	Left Alpha String	Text in Alpha, length in X	Sub-string text in Alpha
54	LOW\$	Lower Case Text	Text in Alpha	Text changed to lower case chars
55	MID\$	Middle Alpha String	Text in Alpha, length in X, start in Y	Sub-string text in Alpha
56	POSTSP	Post-space string	Text in Alpha	Removes all before space char
57	PRESP	pre-space string	Text in Alpha	Removes all after space char
58	RADEL	Right Alpha Delete	Text in Alpha	rightmost character deleted
59	RATOX	Right Alpha to X	Text in Alpha	rightmost character deleted
60	RIGHT\$	Right Alpha string	Text in Alpha, length in X	sub-string text in Alpha
61	ST<>A	Stack and Alpha swap	Stack and Alpha contents	Contents exchanged
62	UPR\$	Upper Case text	Text in Alpha	Converted Text in Alpha
63	USWAP	Unit Swap	"TXT1-TXT2" string in Alpha	String swapped around "-"
64	VIEWA	View Alpha	Text in Alpha	Shows text, doesn't stop
65	VMANT	View Mantissa	Number in X	Shows mantissa
66	XTOAL	char to left Alpha	chr value in X	Prefix Alpha w/ chr
67	-HEPAXA2	Section Header	n/a	n/a
68	AND	X and Y	Numbers in X and Y	Result in X, stack drops
69	PGCAT	Block Catalog	none	Lists block contents
70	CLRAM	Clears Page	Page# in X	Page contents cleared
71	CODE	Codes Hex string	String in Alpha	NNN in X
72	COPYROM	Copies Pages	Pg# in Y (from) and X(to)	FROM Page copied to TO page
73	DECODE	Decodes NNN	NNN in X	Decodes string in Alpha
74	DECODYX	Decodes a number of nibbles	NNN in X, #nibbles in Y	Decoded string in Alpha
75	HEPCHN	Sets HEPAX Ram Chain	Pages as string in Alpha	Chain re-set
76	HEPCHN?	Shows current HEPRAM chain	none	Shows string in Alpha
77	NOT	Not X	Number in X	Result in X
78	OR	X or Y	Numbers in X and Y	Result in X, stack drops
79	RLSRAM	Release RAM	pag# in X	Releases page from HepRAM
80	ROTXY	Rotates Y register X bits	Number in Y, #bits in X	result in X, stack drops
81	SHIFTYX	Shifts Y register X bits	Number in Y, #bits in X	result in X, stack drops
82	XOR	Logical X exclusive-or Y	Number in X	result in X, stack drops
83	X+Y	Bitwise addition	Numbers in X and Y	result in X, stack drops
84	X-\$	Converts X to Alpha string	Number in X	result in X
85	Y-X	Bit-wise subtraction	Numbers in X and Y	result in X, stack drops
86	FCAT2	FAT-2 Catalog	none	Lists sub-functions
87	REV	Shows Revision String	none	Revision shown in LCD
88	NXT	Increases Address string	"ABC>XYZ" in Alpha	Increases ABC+1, XYZ+1

(*) Orange background denotes Launcher functions.

(**) Pink background: Gateways to bank-switched FATs

(***) Blue background denote new or improved in Revisions H to J.

(*IV) Red font denotes new in PWRCL_EXT

Deserving special attention, the function launchers will be discussed later on with the appropriate level of detail. A short write-up article on the Sub-function groups and the multiple FATs is posted in the HP Museum forum, at the following location:

<http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/articles.cgi?read=1195>

Note: Make sure that revision "M" (or higher) of the Library#4 is installed.



© Photo by Geoff Quickfall, 2011

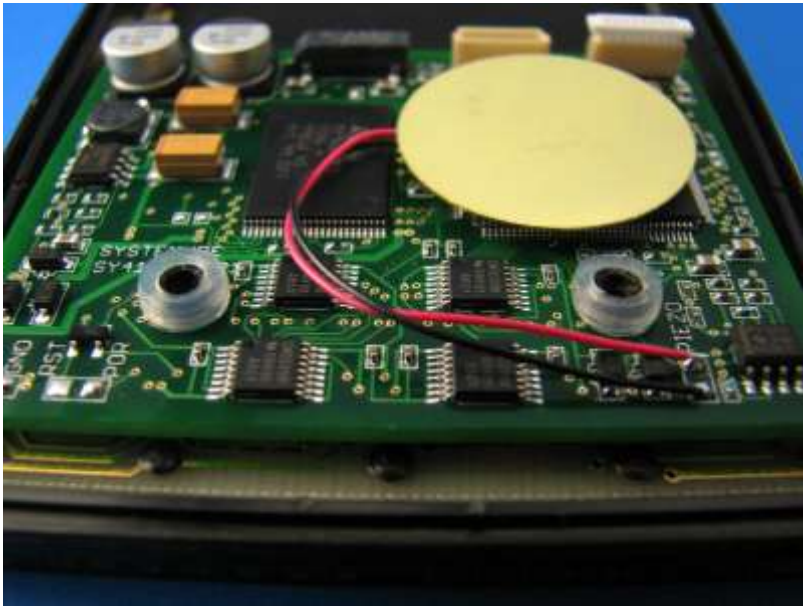
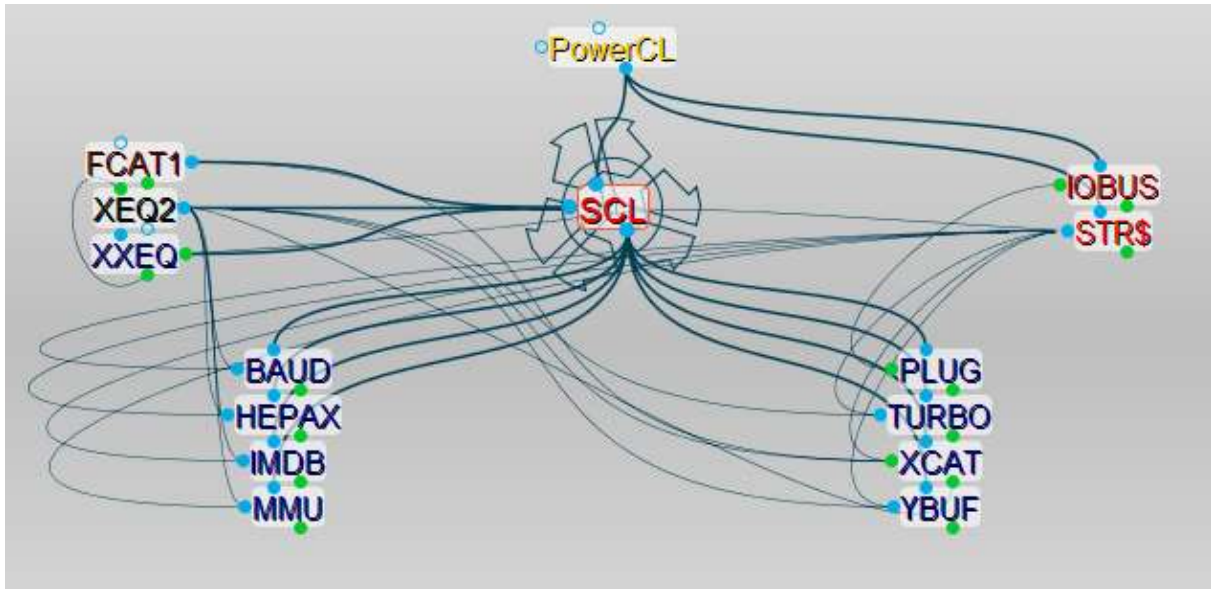
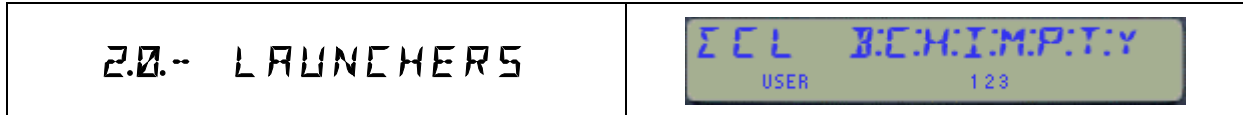


Photo courtesy of Geoff Quickfall. © 2011

2. The functions in detail.

The following sections of this document describe the usage and utilization of the functions included in the POWER_CL module. While some are very intuitive to use, others require a little elaboration as to their input parameters or control options, which should be covered here.



2.0 Function Launchers.

The table below lists the launchers by function groups:

Index	Function	Warnings	Description
1	BAUD	None	Prompts for baud rate setting, plus SERINI, YIMP and YEXP
2	MMU	Light	Prompts for options to manage the MMU configuration
3	TURBO	None	Prompts for CPU speed values, plus TURBO?
4	PLUG"	Light	Prompts for menu of choices, including pg#4 options
5	IMDB	Medium	Calls the corresponding IMDB function
6	HEPX	Medium	Configuration loading for HEPAX set-ups
7	XCAT	None	Extended CATalogs: Buffers, Blocks, CLLIB, etc.
8	YMEM	Medium	Calls the corresponding Expanded Memory function
9	XXEQ	None	Extended Execution Launcher – for both FAT1 and FAT1
10	IOBUS	None	I/O Bus info and control. Has two "screens".
11	SRT\$	None	String and ALPHA Manipulation – Has two "screens".
12	ΣCL	None	Launcher of Launchers -> invokes any of the above

Once you assign **ΣCL** to any key, that alone will give you access to more than 150 functions (!) from that single key – an effective way to make it compatible with other existing key-assignments, saving memory (KA registers) and time. So go ahead and get comfortable with that arrangement as your standard baseline.

Note that the **ΣCL function is designed to work best when assigned to the **Σ+** key.**

Launchers **IOBUS** and **STR\$** were added in revision "J". Each of them has two pages; you can use the **[SHIFT]** key to toggle between them. **STR\$** is triggered by the **[A]** key, whereas **IOBUS** is triggered by the **[SHIFT]** key – use those hot keys to access them from any other launcher prompt.

Prompting functions use a technique called partial key entry, dividing the data entry in two (or more) steps. The keyboard is also redefined, in that only those keys corresponding to the appropriate options are active. The cues in the prompt will offer you indication of which keys are active on the keyboard, and typically are intuitive enough to figure out in each case.

See below a few examples of the prompts, where the choices are to a large extent self-explanatory:



The TURBO options are as follows: none, 2x, 5x, 10x, 20x, 50x, and ?. Use "0" for 20x and the **[T]** key for "50" - as **[2]** and **[5]** are already taken for 2x and 5x speeds.

In general all launchers behave in a similar manner:

- The Back Arrow key either cancels out entirely or removes partial entries;
- Non-active keys will blink the display and maintain the prompt
- The **[A]** key from the main prompt will trigger the **STR\$** launcher
- The **[A]** key from any secondary launcher will revert back to the main launcher, **[ΣCL]**
- The **[ALPHA]** key triggers the **XQ2** function, to execute a sub-function by name
- The **[USER]** key triggers the **XFAT** function, to execute a sub-function by index
- The **[SHIFT]** key triggers the **IOBUS** launcher, or will toggle the "pages" within multi-page launchers
- Holding down the last key briefly shows the invoked function name – for visual feedback.
- This will be followed by "NULL" if kept depressed long enough – last chance to bail out.
- Launchers are not programmable per-se – but *they can be used in PRGM mode* to enter the called-upon function in a program line.

Some of the functions called by the prompts are from the YFNX module, but many others are from the PWRCL_EXT itself as well. The list is long and somehow warps around itself, as launchers are interconnected in a (more or less) logical way.

There are two other "hidden" launchers, not shown in the prompt. They are for the Sub-function Catalogs, **FCAT(1)** and **(2)** – accessed by pressing ENTER^ directly from the **[ΣCL]** and the **XXEQ** launchers respectively -. More about these two later on as well.

Plugging and Unplugging Modules.

The **PLUG#** launcher offers a large selection of choices in its prompts – and therefore deserve special consideration; starting with a review of the internal and external ports and pages on the 41 – as seen in the picture below:

Port 1 <i>Upper Page (9-hex)</i> <hr style="border-top: 1px dashed black;"/> <i>Lower Page (8-hex)</i>	Port 2 <i>Upper Page (B-hex)</i> <hr style="border-top: 1px dashed black;"/> <i>Lower Page (A-hex)</i>
Port 3 <i>Upper Page (D-hex)</i> <hr style="border-top: 1px dashed black;"/> <i>Lower Page (C-hex)</i>	Port 4 <i>Upper Page (F-hex)</i> <hr style="border-top: 1px dashed black;"/> <i>Lower Page (E-hex)</i>

Valid entries for the prompt are shown below – note that **[SHIFT]** toggles between **PLUG** and **UPLUG** actions, and that the prompt choices vary depending on which mode you're in:



- The **[P]** key invokes the standard **PLUG** function in the YFNX– by itself a vastly enhanced version over those in the FYNS and FNYP modules. For starters, it is a prompting function, and it now works on pages as opposed to ports – thus modules are plugged as per their size, starting at the port provided at the prompt.
- Key **[\$]** (same as [P]) and **[I]** keys invoke the **UPLG\$** and **UPLGID** functions respectively, which will unplug a module using its signature string or its IMDB mnemonic – in both cases irrespective of its currently mapped port. – see later descriptions for details.
- Key **[G]** triggers functions **PLUGG** and **UPLGG** - depending on from which one of the above it had been pressed. You may think of these as shortcuts for the "official" **PLUG** function, but using the ALPHA register to hold the ROM id# for **PLUGG**, or "EMPT" for **UPLGG**. Irrespective of how you got to one of them, pressing [SHIFT] toggles between them - the same as it happens with their precursors launchers.
- Keys **[L]** and **[U]** trigger the **LOCK** and **UNLOCK** functions from the YFNX module. Use them to manage the write-to protection of the page, which number (in hex) is entered in the function prompt. Note that you cannot unlock the page where YFNX is configured no matter how many times you use **UNLOCK**, as such protection is connected to the calculator polling points, effectively canceling your action.
- Key **[4]** invokes the page-4 management functions, PPG#4 and UPLG4. They give you a convenient way to configure those special modules that plug on page#4, like the Service ROM, the FORTH module and the LaitRAM. Note however that the Library#4 is not one of them, as it needs to be already configured for the PWRCL_EXT to work.
- Key **[?]** will call functions **PLUGG?** (in the un-shifted mode) and **LOCK?** (in the shifted mode). The former is a quick method to find out the module plugged into a page, simpler than the standard approach using PLUG and the question mark in the prompt syntax. See the YFNX manual for details on **LOCK?**

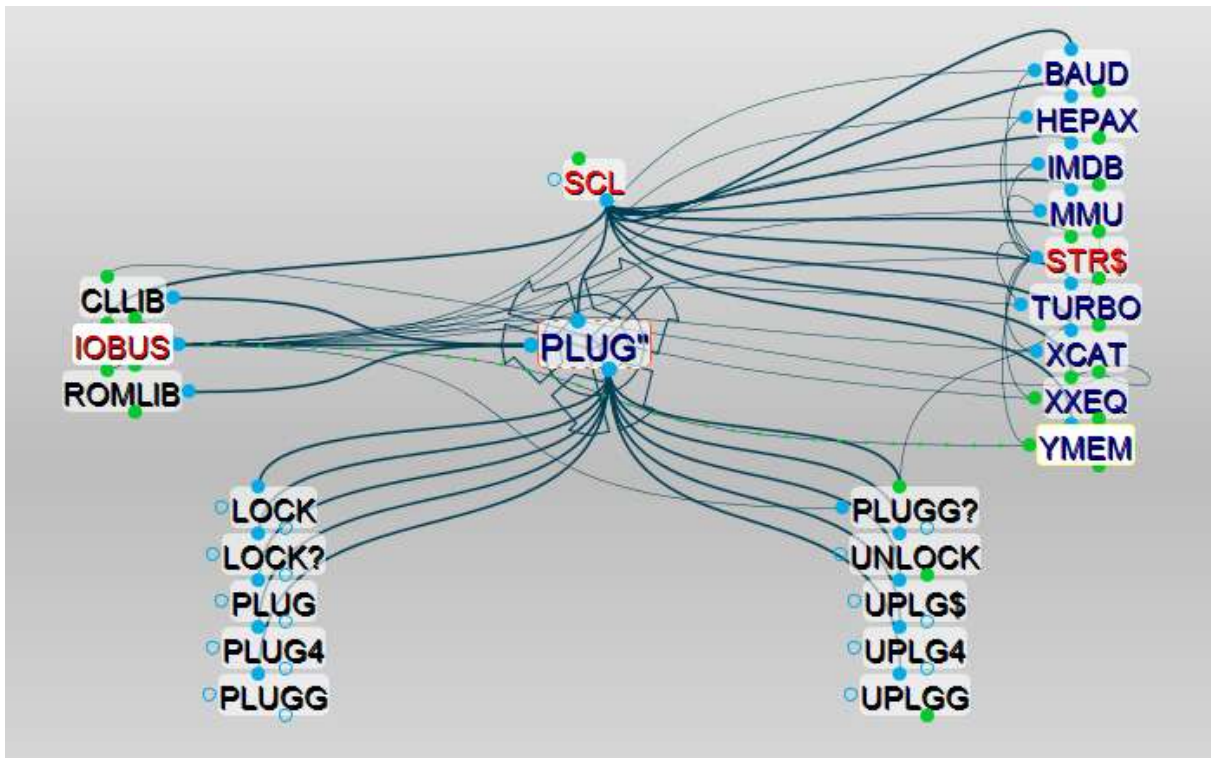
Remarks.- Exercise extra caution with pages #6 and #7, as those locations may be used by system extensions like Printer or HP-IL. Page #6 in particular has more strict demands on the ROM layout that makes it non-suitable for the majority of ROMS.

Note that pages #6 and #7 on **V2 version** of the CL board don't support bank switching; thus they unfortunately aren't a good place for the HEPAX ROM for V2 systems – but makes it the ideal location for V3 boards, where said restriction was removed.

Note: **PLUG, PLUGG and PLGG?** will also allow "4" as valid input, which is nice but potentially very dangerous. In that instance the dedicated functions are not invoked – and the page#4 is used as any other one. Be careful not to inadvertently plug any module not specifically designed for it!

For example the sequence {**PLUG#**, **[SHIFT]**, **"G"**, **4** } will unplug your Library#4, rendering the PWRCL_EXT useless and potentially locking up your calculator. At that point you'd need to plug it back using **YPOKE** or the YFNX function **PLUG** – but not the PWRCL_EXT launcher **PLUG#**

The graphic below summarizes the **PLUG / LOCK** functionality – note also the relationships with the other "sibling" launchers (on the right), its "parent" main launcher, and the related functions (on the left).



Refer to section 3.3.3 (starting on page 37) for further discussions of the page plug/unplug functions, including a few examples of utilization.

Warning: Remember that plugging a module into the "wrong" port location can create minor issues (or major havoc) if you're overwriting some/part of the machine's configuration. A good example is overwriting YFNX itself, or a HEPAX RAM block. Always make sure the destination is safe – using **PGCAT**, the standard CAT2 or better yet the CCD CAT'2.

LASTF: The "Last Function" – at last!

The latest release of the PowerCL module includes support for the "LASTF" functionality. This is a handy choice for repeat executions of the same function (i.e. to execute again the last-executed function), without having to type its name or navigate the different launchers to access it.

The implementation is not universal – it only covers functions invoked using the dedicated launchers, but not those called using the mainframe **XEQ** function. It does however support three scenarios:

- Functions in the module main FATs
- Functions on any other module when called using **XQ1/XQ2**
- Sub-functions from the auxiliary FATs.*

Because functions from the latter group cannot be assigned to a key in the user keyboard, the LASTF solution is especially useful in this case. The following table summarizes the launchers that have this feature:

Module	Launchers	LASTF Method
PowerCL_4BS revision "N"	ΣCL , XCAT, TURBO, BAUD, MMU, IMDB, HEPX, YMEM, XREQ, PLUG	Captures (sub)fnc id#
	IOBUS, IOPG#, STR\$, ALP\$	Captures (sub)fnc id#
	XQ1 _ , XQ2 _	Captures (sub)fnc <i>NAME</i>
	XFAT _ _ : _ _	Captures (sub)fnc id#
	FCAT (XEQ)	Captures (sub)fnc id#

For consistency sake, all prompts connect to the Alphabetical sub-function launcher **XQ2** using the ALPHA key, and with the index sub-function launcher **XFAT** using the USER key.

Note as well that the Alphabetical launchers **XQ1**, **XQ2**, will switch to ALPHA mode automatically. (i.e. there's no need to press ALPHA to start spelling the function name).

Spelling the function name is terminated pressing ALPHA, which will either execute the function (in RUN mode) or enter it using two program steps in PRGM mode by means of the **XQ1/2** plus the corresponding index (using the so-called non-merged approach). This conversion is made automatically – refer to next chapter for more information.

Another new enhancement is the displaying of the sub-function names when invoked via the index-based launcher **XFAT** - which provides visual feedback that the chosen function is the intended one (or not). This feature is active in RUN mode, when entering it into a program, and *when single-stepping the program execution* – but obviously not so during the automated run.

Finally, as of revision "O2" the execution of a sub-function from within the **FCAT** enumeration (using the XEQ hot-key) also supports the LASTF operation for additional convenience. This includes both FCAT1 and FCAT2.

LASTF Operating Instructions

There is no stand-alone LASTF function - the Last Function feature is triggered by pressing the radix key (decimal point - the same key used by LastX) at the launcher prompts. This is consistently implemented across all launchers supporting the functionality in the three modules – they all work the same way.

When this feature is invoked, it first shows "LASTF" briefly in the display, quickly followed by the last-function name. Keeping the key depressed for a while shows "NULL" and cancels the action. In RUN mode the function is executed, and in PRGM mode it's added as a program step if programmable, or directly executed if not programmable.

The functionality works in RUN and PRGM modes – entering the last function (or sub-function) as a program line in PRG mode.

Error Handling.

If no last-function information yet exists, the warning message "NO LASTF" is shown. If the buffer #9 is not present, the error message is "NO BUF" instead.



LASTF Implementation details.

The Last Function functionality is a two-step process: a first one unattended to capture the function id# when it is executed, and a second one triggered by the user's action that retrieves it, shows the function name, and finally parses it for re-execution. All launchers in the PowerCL and other advanced modules have been enhanced to store the appropriate function information (either index codes or full names) in registers within a dedicated buffer (with id# = 9).

The buffer is maintained automatically by the module (created if not present when the calculator is switched ON), and its contents are preserved while it is turned off (during "deep sleep"). No user interaction is required.


There is one register in the buffer reserved for each of the modules featuring this functionality (PowerCL, SandMath, and SandMatrix) – plus the buffer header register also stores the 41Z last-function (which is always in the main FAT). Therefore up to four last-functions can be simultaneously available. A total of five registers are used, as follows:

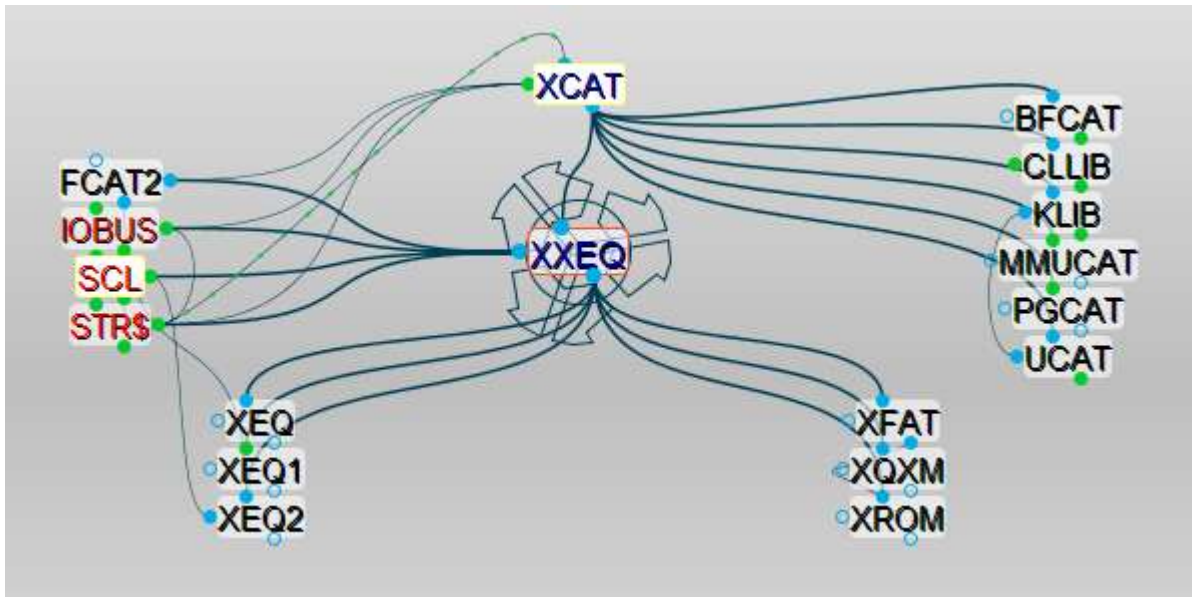
Register	Used for
b4	SandMatrix fcn id# or name
b3	SandMath fcn id# or name
b2	PowerCL fcn id# or name
b1	Time-based Seed (RND)
b0	Buffer Header – w/41z fcn id#

The stored format of the function id# can be either a string of alpha characters (stored in reversed order) representing the function name, or a three-digit hex value representing the sub-function index. In the latter case the [MS] field is marked with an "F" to tell the cases apart.

When the **LASTF** action is triggered pressing the Radix key, the code seeks for the function data in the relevant buffer register, depending on the module carrying the action. When found, it displays the sub-function name (either by mirror-imaging the id# or by looking it up in the corresponding auxiliary FAT. Finally it's parsed to the corresponding section responsible for the execution of the sub-functions.

The LASTF and function name displaying facility together immensely facilitates dealing with sub-functions, almost as if there were standard functions from the main FAT. Go ahead and try it - it's quite something to behold!

<h2 style="margin: 0;">2.1.- EXTENDED XEQ</h2>	
--	--



2.1 Extended Execution.

Pressing [XEQ] at the main [SCL] prompt (even if not shown in the cue) will connect with this launcher, introduced in revision 4B of the PowerCL. It gathers all relevant functions to access any of the sub-functions, as well as a few other tricks for advanced applications.

The functions included in the XEQ launcher are as follows:

Key	Function	Warnings	Description
0	XEQ	None	Invokes the native XEQ function
1	XQ1	None	Calls XQ1, for FAT-1 sub-functions calling by NAME
2	XQ2	None	Calls XQ2, for ANY sub-function calling by NAME
F	XFAT	None	Calls XFAT, for ANY sub-function calling by index
R	XROM	None	Calls XROM, to execute any ROM function (if present)
Q	XQ>XR	None	Converts XEQ instructions to XROM
M	XQXM	None	Calls XQXM, to execute a program File

Without a question this is one of the most intriguing parts of the POWERCL. Perhaps a quick refresher on the theory of operation of sub-functions is appropriate, so let's get on with it:-

- Sub-functions can be called by their name (using an ALPHA prompt) or by their FAT index (using a numerical prompt). So in principle two main functions are needed per each FAT and since there are two of these FAT groups, it'd be logical to assume that four functions would be needed in total, one per each type times two FATs.
- To reduce the requirements on the main FAT, the two numeric launchers have been consolidated into a single function (XFAT __: __) with two prompt fields: use the first for the FAT, and the second for the index.

- Furthermore, having two different Alpha launchers may be confusing; as the user would need to remember the group any of the sub-functions belongs to. To avoid this, both launchers are *capable of searching for any sub-function, and not only for those in its own FAT.*
- But that's not all – after a while the user won't necessarily remember whether a given function of the module is either in the main FAT-0, or in any of the two sub-function groups. To avoid this issue the **XQ1** and **XQ2** launchers are also capable of searching in the main FAT, and will do so if the name hasn't been found in FAT-1 and in FAT-2.

In fact they will also search in all main FATs present in the system (which may have more modules plugged in), and not only the POWERCL one. So **XQ1** and **XQ2** are **universal XROM function caller – by name**.

Note however that they won't find the mainframe functions (like BEEP, SIN, etc.). If that's what you're after you can also use the option "0" provided in the **XXEQ** launcher to invoke the native **XEQ** – in all its glory: ALPHA, numeric, **IND**irect, etc.

The above considerations would suggest that having two launchers with the same capabilities is somehow redundant, and sure you'd wonder why not removing one of them, freeing up one more entry in the main FAT for other purposes. There is however an important reason to have it there: it is needed to have the programming capability for the sub-functions in each FAT; let's explain.

When **XQ1/2** are used in program mode to call a sub-function by its name, it will create two program lines. The first will be either **XQ2** or **XQ1**, depending on which FAT the sub-function belongs to. The second will be the index within the FAT.-

Therefore "**XQ1_nn**" is different from "**XQ2_nn**", even for the same "nn" – and this is exactly the reason why **XQ1** and **XQ2** both need to be present. But even if both must be there, the user really only needs to use one – as the conversion from the spelled-out names will occur automatically on a need-to basis!

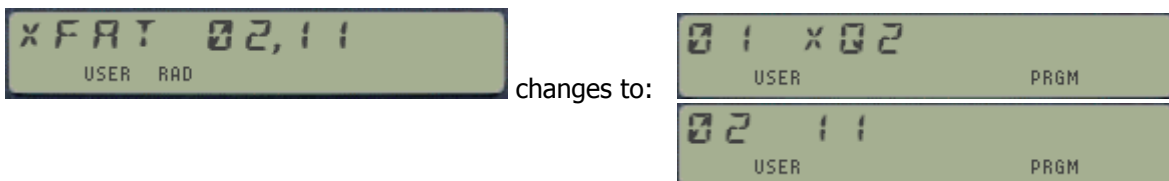
Let's see an example. Say we want to execute function **FLTYPE** to find out the type of the X-Mem file which name is in ALPHA. This can be done either by its name or by its index. In the first case we'd type:

XQ1 (or **XQ2**), **F,L,T,Y,P,E**,[ALPHA] (note how ALPHA will come up automatically)

If the numeric launcher is to be used instead, the key entry sequence will be:

XFAT 02, 11; or also: **XXEQ, [F], 02, 11**

Either way this creates the same two program lines in PRGM mode:



Note as well how [the sub-function NAME is briefly displayed](#) during this process – providing visual feedback on the selected function - even if the name wasn't used to enter it.

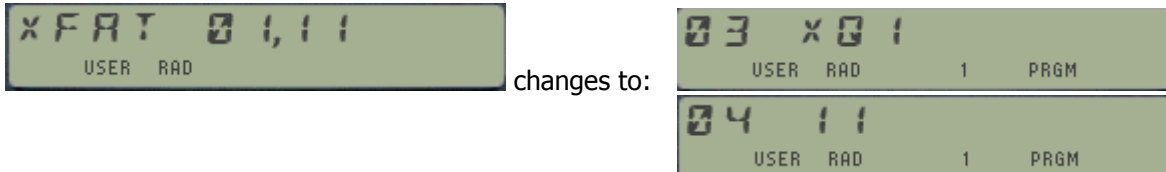
Let's now see how to execute function **YBK-XM**, which is a sub-function within FAT-1. If done by name the user could press –now using the **XXEQ** launcher from the scratch

XXEQ, 1, Y,B,K,-,X,M, [ALPHA] or
XXEQ, 2, Y,B,K,-,X,M, [ALPHA] indistinctly.

And if the numeric launcher were to be used instead, the entry sequence would be:

XFAT 01, 11; or also: **XXEQ, [F], 01, 11**

Any of these three options creates the same two program lines in PRGM mode:



XFAT is clever enough to know that the only valid inputs for the first prompt are 0,1, or 2. Should you enter any other number the calculator will politely ignore the input and continue to display the prompt – much more user friendly than allowing the entry and then coming back with a "DATA ERROR" message.

And speaking of this, the function called for entry value of zero is **XROM** – a jewel of a routine originally written by Clifford Stern – which in turn prompts for the XROM id# and the function number for a function from any ROM. All in programmable splendor and ready to execute!

The remaining functions **XROM** and **XQXM** are both covered in a later section of the manual. They are included in the **XXEQ** launcher for additional convenience. Nothing short of amazing if you ask me – a great implementation made even better with the addition of the FAT Catalogs, described below.-

Sub-function CATALOGs

One of the missing features in the original HEPAX implementation of the Multi-function groups (**XFA/XFA** and **HEPAX/HEPAXA**) was the lack of a sub-function catalog. Having such enumeration option would certainly be convenient, and very useful if it also had the capability to directly launch the function which name is being displayed.

But not so anymore! - Meet **FCAT1** and **FCAT2**, the sub-functions CATALOG facility that will allow you do exactly that. To invoke it you can use its name in the **XQ1** or **XQ2** ALPHA prompts, or its index in the numeric prompts **XFAT 01, _ _** & **XFAT 02, _ _**

No doubt you surely have noticed (or are about to) that the sub-functions are arranged in alphabetical order, all except **FCATn** which is placed at the bottom of each list. Besides, the first function (the section header) can also be used to trigger **FCATn** – *so that you don't really need to know their respective indices* – (which happen to be 89 in FAT-1 and 86 in FAT-2); and can use "00" in both cases as an enhanced usability shortcut.

The following hot keys are available:

- [R/S] to stop/resume the sequential enumeration;
- [SST]/[BST] to single-step forward or backwards while in manual mode;
- [ENTER^] to jump to the next/previous section within the catalog;
- [XEQ] to execute the function being displayed – *now supports LASTF*;
- [SHIFT] to flag the forwards/backwards direction;
- [←] Back arrow will terminate the enumeration; and
- [ON] to power the calculator OFF.

This is almost identical in form and feature to the CCD approach for the system CATalogs – all except the [A] hot-key for the assignment. – It should be noted that sub-functions cannot be assigned to a key on the keyboard, and therefore the absence of such an option.

Note also that the enumeration will be output to a printer when NORM/TRACE is selected, and a forwards enumeration is performed - for a hard-copy record of the listing.

Sections within the two sub-FATs

Each multi-function group has a lot of sub-functions. Navigating the catalogs could be very tedious without the existence of sub-sections within them, and the capability to jump from section to section.

The sections underneath the sub-FATs are listed below:

For FAT-1

- YFNX NEW including the Expanded memory fncs. plus more.
- PAGE FNS including all page-related and I/O Bus function
- XTRA FNS with a long list of utilities and MCODE-related functions
- 56 BITS with digital functions (on the complete register size).

For FAT-2

- PWREXT B3 with the Buffer and Extended Memory enhancements
- ALPHA/DSP for the ALPHA and Display functions, and
- HEPAXA2 including the functions from the HEPAX group, plus some more.

Hotkey [ENTER^] will toggle between the sections or the individual functions listing, exactly the same as it works in the CCD –style catalogs. This makes it easier to navigate your way in such long lists, considering that FAT-1 has 89 functions and FAT-2 87 and otherwise waiting for your target sub-function to appear in the enumeration could be a relatively long wait.

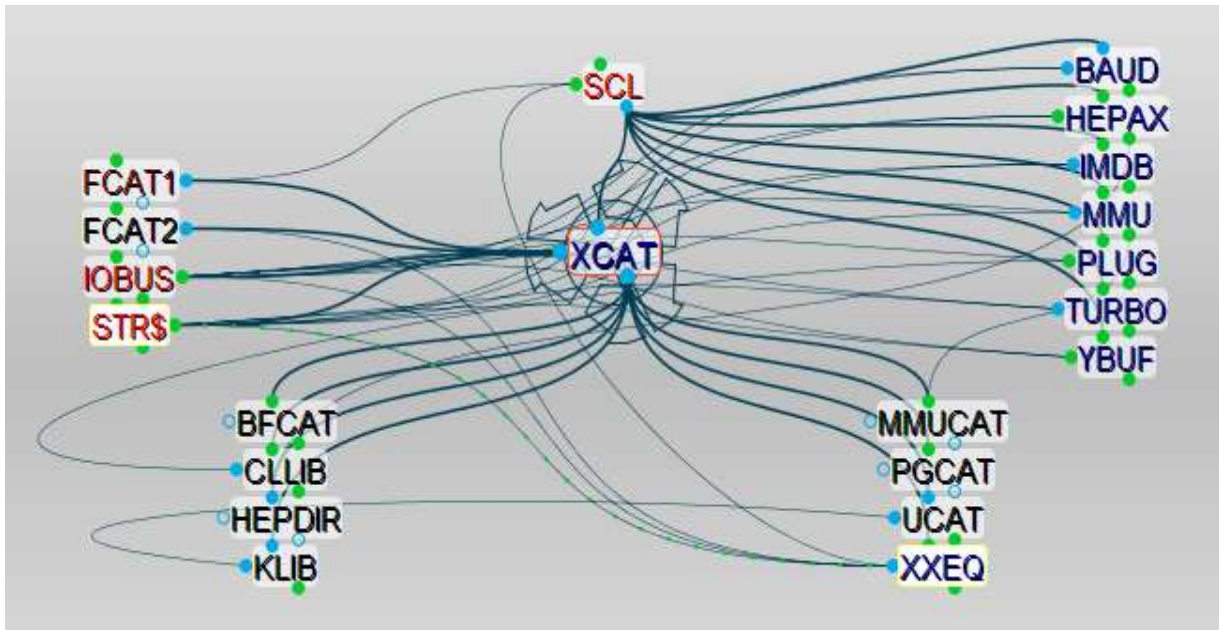
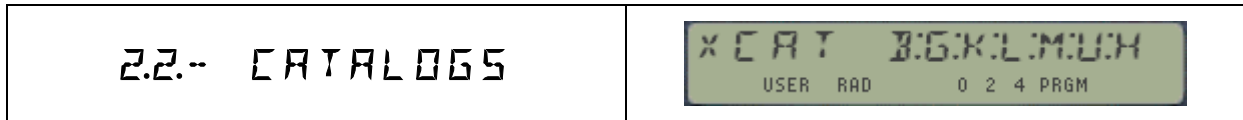
Note: The following shortcuts are the most-convenient ways to access the sub-CATalogs:

- Pressing ENTER^ at the XCL prompt will invoke the sub-function Catalog for **FAT-1**,
- Pressing ENTER^ at the XXEQ prompt will invoke the sub-function Catalog for **FAT-2**.

A new feature in revision "K" allows an even easier approach – by means of the XCAT function described in the next section:

- XCAT, "1" will invoke **FCAT1**, the sub-function Catalog for **FAT-1**,
- XCAT, "2" will invoke **FCAT2**, the sub-function Catalog for **FAT-2**,

Several ways to perform the same task – but all of them intuitively accessible from the expected location to provide choices to power users.



2.2. CATalogs and Catalogues...

The additional CATalogs are accessed by **XCAT**, and include the following:

key	Function	Warnings	Description
G	PGCAT	None	Borrowed from the HEPAX ROM – shows the 4k-blocks contents.
B	BFCAT	Light	Lists those elusive buffers present in the system.
M	MMUCAT	None	Lists the MMU mappings into each block.
H	HEPDIR	None	Calls the HEPAX' HEPDIR function
K	KLIB_	None	Constants Library Selections
L	CLLIB_	Light	CL ROM Image Library - alphabetical or by type.
U	UCAT_	None	Lists all available units for UMS conversion.
X	XSEQ	None	Another entry for the Extended XEQ launcher
1	FCAT(1)	None	Easiest way to access functions in FAT-1
2	FCAT(2)	None	Easiest way to access functions in FAT-2

If you're like me you'll like to have good visibility into your machine's configuration. With its ROM Library and MMU settings the CL adds a few dimensions to the already rich 41CX system – and the goal is to have equivalent catalogue functions to review the status and options available.

Each CATalog has its own idiosyncrasies, but in general they feature single-step modes, and have "hot keys" to allow for specific actions – like deletion of buffers, navigation shortcuts, and direct plugging of ROMs into a port. This makes chores like searching for the correct syntax and plugging a module from the library a trivial task.

Some (**PGCAT**, **BFCAT**, **KLIB**, **UCAT**) are not strictly related to the CL, and will also work on a standard 41. Obviously **MMUCAT** is only meaningful for a CL machine, and will return zeroes if the CL board is not installed.

CATalog functions are notoriously complex and take up a significant amount of space – yet you'd hopefully agree with me that the usability enhancements they provide make them worthwhile the admission price.

2.2.1. Buffer CATALOG

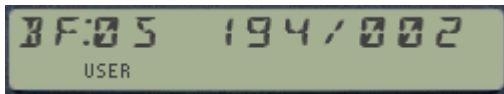
BFCAT	Buffer CATalog	Hot keys: R/S, SST, SHIFT, D, H	
[D]	Deletes Buffer	<i>In manual mode</i>	Asks Y/N?
[H]	Decodes Header register	<i>In manual mode</i>	

This function is very close to my heart, both because it was a bear to put together and because the final result is very useful and informative. It doesn't require any input parameter, and runs sequentially through all buffers present in the calculator, providing information with buffer id# and size.

41 buffers are an elusive construct that is mainly used for I/O purposes. Some modules reserve a memory area right above the KA registers for their own use, not part of the data registers or program memory either. The OS will recognize those buffers and allow them to exist and be managed by the "owner" module – which is responsible to claim for it every time the calculator is switched on.

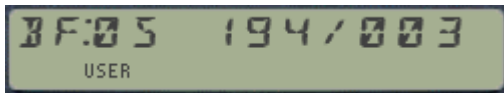
Each buffer has an id# number, ranging from 1 to 14. Only one buffer of a given id# can exist, thus the maximum number present at a given time is 14 buffers – assuming such hoarding modules would exist – which thankfully they don't.

For instance, plug the AOSX module into any available port. Then type **PI, SEED**, followed by **BFCAT** to see that a 2-register buffer now exists in the HP-41 I/O area – created by the **SEED** function.



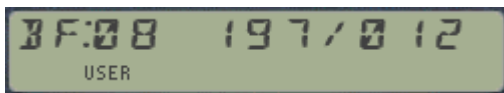
id#=5, buffer at address 194, size=2, properly allocated.

Suppose you also change the default word size to 12 bits, by typing: 12, **WSIZE**. This has the effect of increasing the buffer size in one more register, thus repeating **BFCAT** will show:



id#=5, buffer at address 194, size=3, properly allocated.

Say now that you also plug the 41Z module into a full port of your CL. Just doing that won't create the buffer, but switching the calculator OFF and ON will – or alternatively execute the **-HP 41Z** function. After doing that execute **BFCAT** again, then immediately hit **R/S** to stop the listing of the buffers and move your way up and down the list using **SST** and **BST**. You should also see the line for the 41Z buffer, as follows:



id#=8, buffer at address 197, size=12, properly allocated.

If the module is not present during the CALC_ON event (that's to say it won't re-brand the buffer id#) the 41 OS will mark the buffer space as "reclaimable", which will occur at the moment that PACKING or PACK is performed. So it's possible to have temporary "orphan" buffers, which will show a question mark next to the id# in the display. This is a rather strange occurrence, so most likely won't be shown – but it's there just in case.

Perhaps the best example is the Time module, which uses a dedicated buffer to store the alarms data.

The table below lists some of the well-known buffers that can be found on the system:

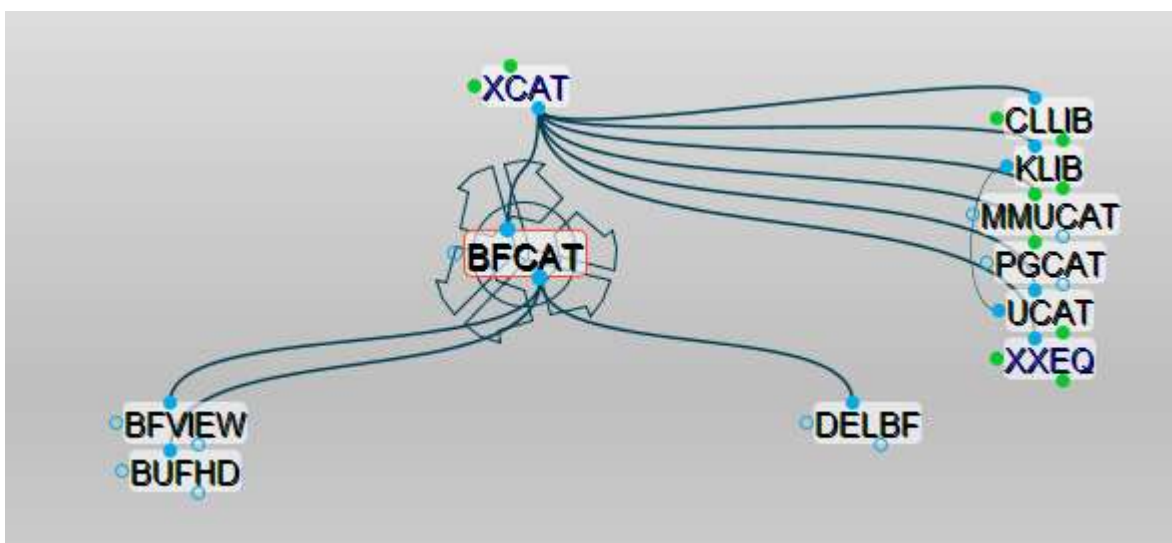
Buffer id#	Module / EPROM	Reason
1	David Assembler	MCODE Labels already existing
2	David Assembler	MCODE Labels referred to
3	Eramco RSU-1B	ASCII data pointers
4	Eramco RSU-1A	Data File pointers
5	CCD Module, Advantage	Seed, Word Size, Matrix Name
6	Extended IL (Skwid)	Accessory ID of current device
7	Extended IL (Skwid)	Printing column number & Width
8	41Z Module	Complex Stack and Mode
9	SandMath, PowerCL; 41Z	Time Seed; Last Function data
10	Time Module	Alarms Information
11	Plotter Module	Data and Barcode parameters
12	IL-Development; CMT-200	IL Buffer and Monitoring
13	CMT-300; FORTH Module	Status Info; FORTH Code
14	Advantage, SandMath	INTEG & SOLVE scratch
15	Mainframe	Key Assignments

BFCAT has a few hot keys to perform the following actions in manual mode:

- [R/S]** stops the automated listing and toggles with the manual mode upon repeat pressings.
- [D]** for instant buffer deletion – there's *no way back, so handle with care!*
- [H]** decodes the buffer header register. Its structure contains the buffer ID#, as well as some other relevant information in the specific fields - all buffer dependent.
- [V]** Views the contents of the buffer, sequentially showing its registers in the display
- [SHIFT]** flags the listing to go backwards – both in manual and auto modes.
- [SST]/[BST]** moves the listing in manual mode, until the end/beginning is reached
- [<-]** Back Arrow to cancel out the process and return to the OS.

Like it's the case with the standard Catalogues, the buffer listing in Auto mode will terminate automatically when the last buffer (or first if running backwards) has been shown. In manual mode the last/first entry will remain shown until you press BackArrow or R/S.

Should buffers not be present, the message *"NO BUFFERS"* will be shown and the catalog will terminate. *Note also that the catalogue will be printed if in NORM/TRACE mode*, producing a record of all buffers present in the system!



2.2.2. Interrogating the MMU.

MMUCAT	MMU CATalogue	No inputs	
ADRID	Gives ROM id# from ADR	Expects string in Alpha	
FYNZ?	FYNS Location Finder	No inputs	
PLUGG?	ROM id# in page by X	Prompts for page#	Valid inputs are 4, 6-F

MMUCAT is really a FOCAL program that drives the function **ADRID**, the real engine behind it – not to be confused with the capital city of a country I know quite well. **ADRID** is obviously programmable. The idea is simple: produce a list of the MMU mappings into the different pages, showing either the ROM id# or the address (Flash or SRAM) currently mapped to the port.

A loop is executed starting on page #3, and up until page #F. Each iteration retrieves (pokes more appropriately) the address written into the corresponding MMU register, then searches it against the internal ROM id# table written in bank-2 of the PWERCL_EXT module. More about this later.

Note that full-port modules will return the ROM id# attached to the lower half, and the Flash address mapped to the upper half. sRAM MMU entries will return the corresponding sRAM address.

While similar to the CAT2 concept, this really has an MMU-oriented perspective of things, and thus is purely a 41 CL feature – it'll render all entries zero if used on a "regular" HP-41. The program listing is rather simple – as **ADRID** does all the weight lifting under the hood:

01	LBL "MMUCAT"		20	LBL 00	
02	"0-0000"		21	"8040"	<i>prefix</i>
03	ASTO X		22	XTOA	
04	52	"4"	23	ARCL Y	<i>page#</i>
05	XEQ 00		24	YPEEK	<i>read MMU rg,</i>
06	CLX		25	YSWAP-	<i>swap around ". "</i>
07*	54,057	"6" to "9"	26	YCL-	<i>delete from ". "</i>
08	LBL 02		27	YBSP	<i>back space</i>
09	XEQ 00		28	ATOX	
10	ISG X	<i>next</i>	29	RDN	
11	GTO 02		30	ADRID	<i>decode address</i>
12	CLX		31	XTOA	
13*	65,07	"A" to "F"	32]-: "	
14	LBL 01		33	-3	
15	XEQ 00		34	AROT	
16	ISG X	<i>next</i>	35	RDN	
17	GTO 01		36	AVIEW	<i>show ID#</i>
18	CLD		37	PSE	<i>pause</i>
19*	RTN		38	END	

A related function is **YFNZ?**, which returns the page number the YFNS is currently plugged in. This can come very handy in your programs to avoid overwriting it with other modules – as we'll see in the HEPAX configuration routines. Note that **YFNZ?** is located in Bank-3, thus requires **XQ2** to launch.

Another related function is **PLUGG?** - It interrogates the MMU to find out which module is plugged into a given page – entered in the prompt. Note that it is programmable, and that in program mode it will take the page# from the X registers. This is all **page**-driven, and not based on the *port* number. There is no restriction in the input to the page number, however the returned values for pages 0,1,2,3, and 5 don't quite have the same meaning.

PLUGG? Also uses **ADRID** to decode the string returned by **YPEEK** – which reads the MMU address mapping the corresponding page. In the **YFNZ?** case there's no need to look up in the ROM id# table since we know what we're looking for – just need to check all pages looking for that specific string.

2.2.3. Page CATALOG

PGCAT	Page Catalog	VM Electronics	Source: HEPAX Module
--------------	---------------------	----------------	----------------------

PGCAT is taken from the HEPAX Module (called **BCAT** there, within the HEPAX sub-functions group) - and written by Steen Petersen. **PGCAT** enumerates the first function of each page, starting with page 3. The enumeration can be stalled pressing any key other than R/S or ON, but the individual functions won't be listed.

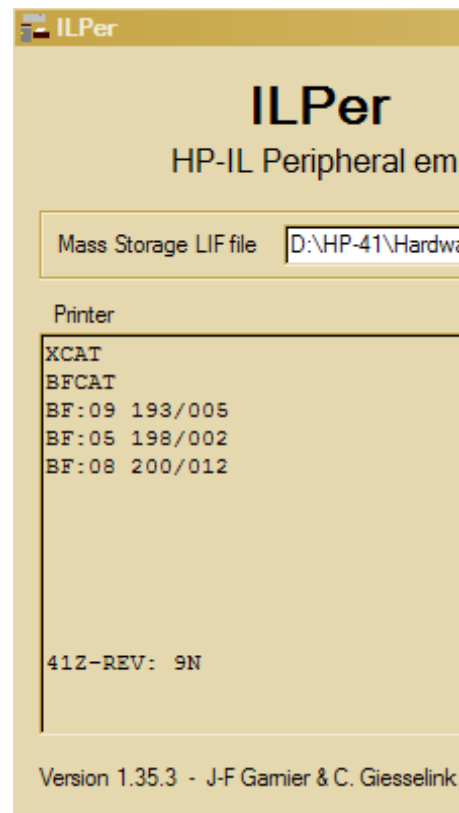
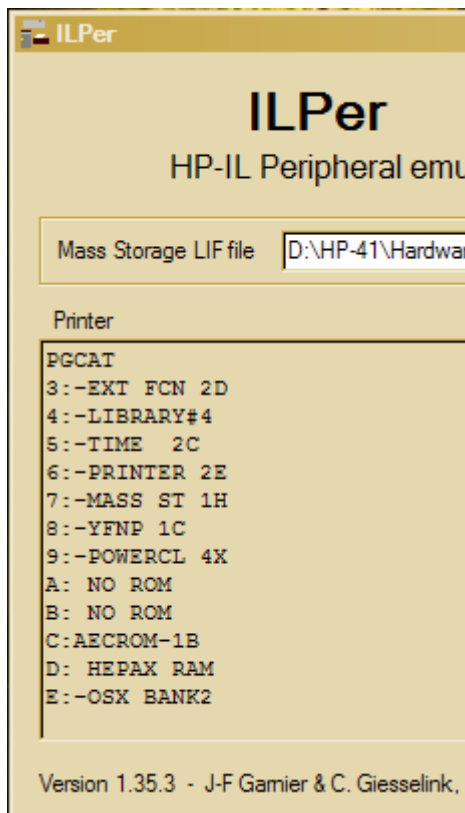


PGCAT Lists the first function of every ROM block (i.e. Page), starting with Page 3 in the 41 CX or Page 5 in the other models (C/CV). The listing will be printed if a printer is connected and user flag 15 is enabled.

- Non-empty pages will show the first function in the FAT, or "NO FAT" if such is the case
- Empty pages will show the "NO ROM" message next to their number.
- Blank RAM pages will also show "NO FAT", indicating their RAM-in-ROM character.

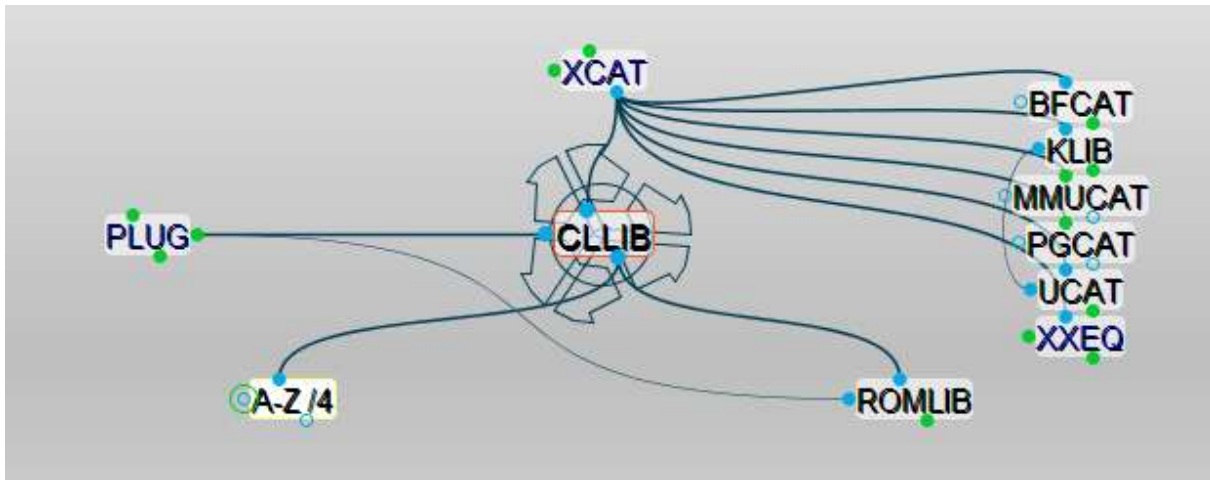
No input values are necessary. This function doesn't have a "manual mode" (using [R/S]) but the displaying sequence will be halted while any key (other than [R/S] or [ON]) is being depressed, resuming its normal speed when it's released again.

See below the printout outputs from both **BFCAT** and **PGCAT** using J-F Garnier's PIL-Box and the ILPER PC program, showing a nice traceability of the pressed keys:



We'll encounter this function again as part of the **IOBUS** / **IOPG#** function launchers later in the manual.

2.2.4. A wealth of a Library – with two access modes.



ROMLIB _	ROM Library by type	Prompts for type: E:F:G:M;S:U;X	Has Hotkeys
CLLIB _	CL Library by name	Prompts for A-Z	Has Hotkeys
[P]	Invokes PLUG _		
[A]	Copies id# shown to Alpha		

One of the most notable features of the CL is its extensive ROM image library, allowing you to plug almost any conceivable module ever made (of which I have contributed a few) into your 41CL just by using one of the **PLUGxx** functions. The input syntax requires that the correct ROM ID string be written in Alpha, and certainly there are a few of those to remember – and rather similar to each other since the string is only 4 characters long.

These two functions come to the rescue – providing listings of all available module mnemonics, either alphabetically or by their type - so you can review them, and –eventually – plug the ROM directly from the catalogue for convenience sake.

ROMLIB prompts for a ROM type, whereas **CLLIB** prompts for an alphabetical section, A to Z. Pressing **[SHIFT]** at this point toggles between both modes, alphabetical or by type. Both catalogues can run in auto mode or can be stopped using R/S, and then the listing can proceed in manual mode using SST and BST as you can expect.

It is in manual mode where you can use the other shortcuts or “hot keys”, as follows:

- **[ENTER^]** skips to the next section (or previous if running backwards)
- **[A]** to copy the id# shown to Alpha
- **[P]** to exit the catalog and invoke the **PLUG_** function launcher
- **[SHIFT]** changes the direction of the listing, backwards <-> forwards
- **[<-]** Back Arrow will cancel out the catalog.

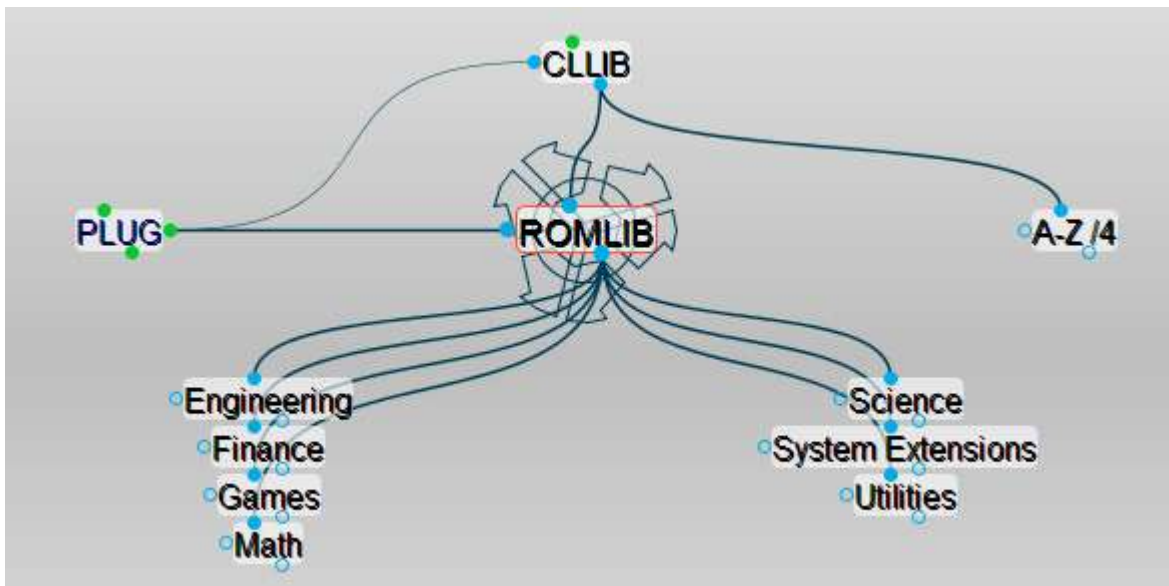
The enumeration terminates in auto mode when the last ROM id# (or first one if running backwards) has been reached. Also keeping any key depressed in RUN mode will halt the sequence displaying until it's released again, so it's easier to keep tabs with the enumeration.

The same considerations made about plugging modules apply here – be careful not to inadvertently overwrite anything you're using with a new ROM image (specially important for YFNS), as there's no check whether the target location is already used or not – with the only exception being function **PLUGG**

As you can guess there is a lot of code sharing between **ADRID** and these two ROM library catalogue functions. Fundamentally they all use a ROM id# table within bank-2 of the POWER_CL ROM to look up for the string, and fetch the address in Flash of the corresponding image. This table is quite long, occupying more than 1k in the ROM – yet worth every byte.

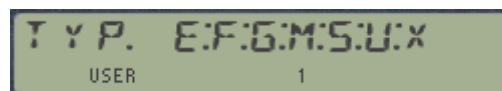
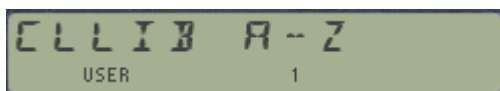
The prompt in **ROMLIB** suggests the different ROM types as follows:

- **[E]** Engineering-related modules in all the main branches: EE, ME, etc.
- **[F]** Finance-related modules – you gotta love those too...
- **[G]** Games and entertainment related modules;
- **[M]** Math-related modules, the heart of the machine for some;
- **[S]** Applied Science-related modules, excluding Engineering;
- **[U]** The Utilities group, all those nice packs adding extra tools to the system;
- **[X]** System-Extensions group, all what makes the 41 much more than just a calculator.
-



As you can see these groupings are somewhat loosely defined, yet it's simple and intuitive enough to have a good handle on the categories – which is the main objective for the searches. Also there's no distinction between HP-made or 3rd. Party modules in this scheme.

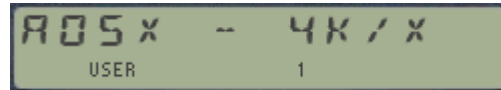
The "A-Z" prompt entry in **CLLIB** is a refinement of the same idea: it provides a handy shortcut to start your search in the appropriate section, so there's no need to review all the preceding ones – which can be very lengthy considering the sheer number of them, even if you used **ENTER**^ to skip sections. The implementation is quite nice, even if it's the author who says it – have a look at the POWERCL_Blueprint if you're curious about the MCODE implementation details.



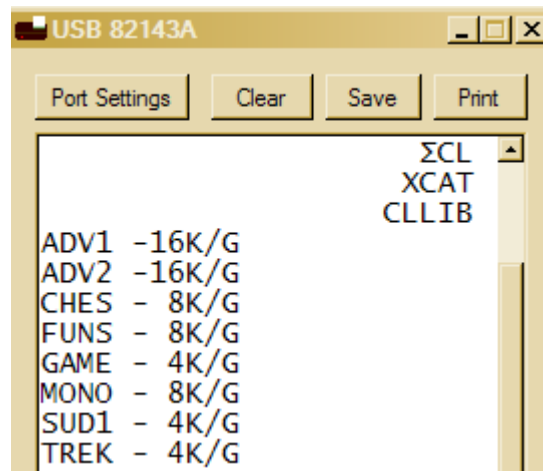
If the section doesn't have any ROM id# starting with such letter (which currently only occurs with the [W] letter) the message "NO MATCH" will be shown. Non-alphabetical keys are not valid entries for **CLLIB**, and will cause the display to just blink and maintain the prompt. Lastly, selecting [X] will list the general-purpose placeholders; refer to the CL manual for details on those.

The enumeration output includes three pieces of information, as follows:

- (1) The ROM ID name (entry used by the CL in the PLUGxx functions),
- (2) Its size (either single or multi-page modules), and
- (3) The type it belongs to according to the classification explained above.



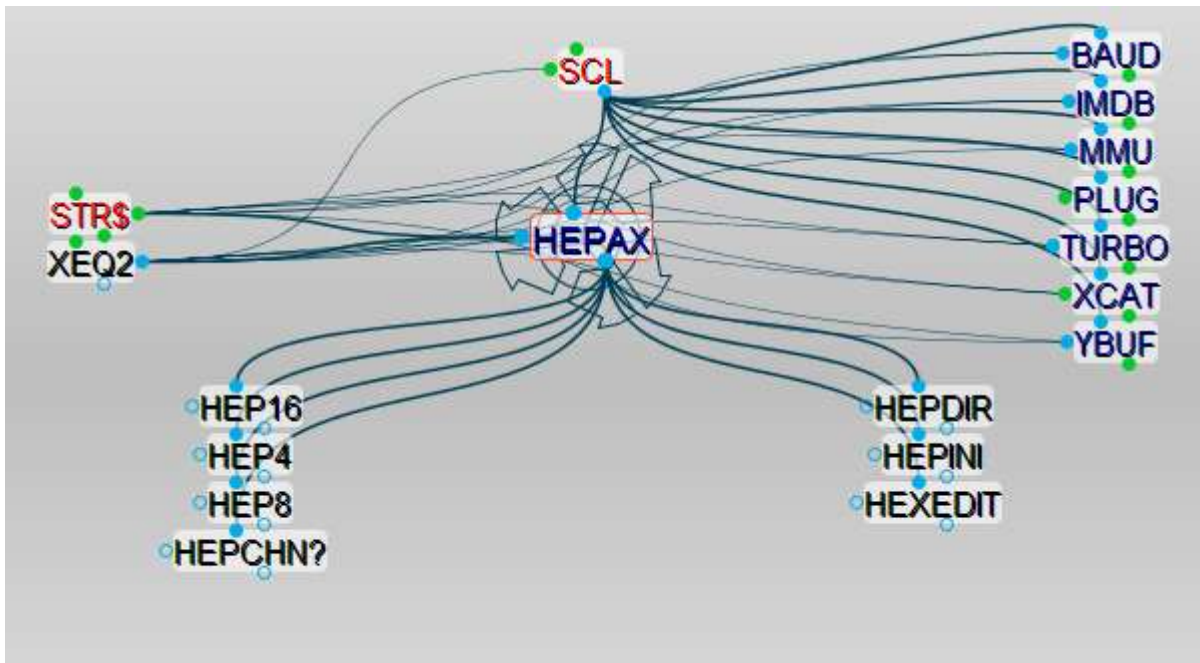
Like it was the case for **BFCAT**, *these two functions fully support the printer TRACE/NORM modes*, generating a complete printout record of the ROM enumeration for your convenience. See appendixes for a complete listing of the module database, and the example below listing just the Finance and Games modules, taken as a direct cut & paste from the output using the USB-41 module:



At any given time the automatic enumeration can be stopped using **[R/S]**, and continued manually in single-step mode using **SST**, either forwards or backwards controlled by the **[SHIFT]** key – yet the entry will only be printed going forwards, to eliminate redundancy. Finally, the hotkeys **[A]** and **[P]** will copy just the ROM CL_ID text into Alpha, without the size/type details so it's ready to be used by any **PLUG#** function.

2.3. HEPAX & SECURITY

HEPX 4:8:6:M:I:3:7
USER 3



2.3.1. Configuring the HEPAX system.

HPX4	4k RAM HEPAX Setup	Prompts for HRAM Page#	ROM into previous
HPX8	8k RAM HEPAX Setup	Prompts for HRAM page range	ROM into previous
HPX16	16k RAM HEPAX Setup	Prompts for HRAM page range	ROM into previous

These three functions will prepare the CL to hold a properly configured HEPAX file system, starting from the scratch. The process can be divided into four distinct parts:

1. First copying the HEPAX RAM template from Flash into the appropriate number of SRAM blocks, as many times as needed depending on the choice.
2. Followed by mapping those SRAM blocks to the 41 ports using the MMU,
3. Configuring the Hepax FileSystem using **HEPINI** - so that they are enabled for the HEPAX ROM to use.
4. Besides that, the functions will also map the HEPX ROM image to the page preceding the first HRAM block, as shown in the table on next page.

They will first present a confirmation prompt - as a precaution to avoid overwriting existing HRAM blocks. Only Y/N input is allowed at this point. If "Yes", they'll prompt for the **first** page to locate the HRAM blocks; in hexadecimal format. Allowed entries for each case are shown in the prompt. Inputs other than those will simply be ignored and the prompt will be maintained.

OK? Y/N
USER 3

"Y":

HPX4 9-F_
USER 3

or:

HPX8 9-E_
USER 3

or:

HPX16 9-C_
USER 3

You must be fully aware that the previous MMU mapping to those ports will be overwritten. **The exception being the YFNS ROM itself** – as the functions will check whether it is currently mapped to any page within the affected range – aborting if that's the case. A nice built-in protection that avoids getting in trouble.



WARNING: Obviously these functions are not to be used frequently, *since each execution will wipe off the existing content of the HRAM pages*, overwritten with blank FLASH templates (!). Therefore the "medium" warning sign, proceed with caution. Before the page prompt, a conformation will be requested for safety, showing the message "OK? Y/N" – you know what to do.

See the appendix 2 for a listing of the FOCAL programs that implement this functionality.

WARNING: because the page# is used for the XROM id#, some conflicts *will* occur when using pages C or F – clashing with YFNS and POWERCLXROM id#'s. Make sure you EDIT those manually.

Re-Initializing the Pages control fields.

HEPX	HEPAX Fns. Launcher	Prompts "4:8:6:H:I:D"	Accessible from ΣCLF
HEPINI	Initializes File System	Prompts for values	Author: Howard Owen

HEPINI is used to initialize the HEPAX File System on the CL. This is needed on the CL because this feature is disabled in the HEPAX ROM image included in the CL Library, thus the addition here. *It also allows for dynamic configuration changes*, modifying the number and location of HRAM pages set up.

In manual mode, the function takes two parameters: **the number of HEPAX RAM pages to configure** and **the address of the first one**. Note that even if the first prompt is a DECIMAL entry, the double quotes will remind you that the second one is in HEX, with valid inputs being 8,9, and A-F.



HEPINI is also programmable. In PRGM mode it takes the number of HRAM pages from Y, and the first page address from X – both in DECIMAL format.

The procedure consists of writing a few control words into strategic locations within each HRAM page, so that the HEPAX will recognize them as being part of its File System. Those locations and byte values are shown in the table below:

Address	Byte value	Comment
x000	ROM id# => equal to the page#	Always done
xFE7	Previous HRAM page id# (zero if first)	Always done
xFE8	Next HRAM page id# (zero if last)	Always done
xFE9	Fixed value = 091	Won't be overwritten if not zero
xFED	Fixed value = 090	Done always
xFEf	Fixed value = 091	Won't be overwritten if not zero
xFF1	Fixed value = 0E5	Done always
xFF2	Fixed value = 00F	Done always
XFF3	Fixed value = 200 (or 100)	Done always

Two of the byte values shown in the table above located at addresses 0xpFE9 and 0xpFEf have a different treatment: they will be branded only if the previous content is zero. They denote the initial

address in the page where the next file or program will be written using **HSAVEP**, **HCRFLAS** and **HCRFLD**. Their values will vary as more programs or content is written to the HRAM page, thus should not be overwritten by **HEPINI** – or else the HEPAX FileSys catalog will become corrupt.

This explains why **HEPINI** won't disturb the actual contents of the HRAM FileSystem, so it can be used at any time provided that the entries used are compatible with the HRAM arrangement. It is also possible to use them to configure only a subset of the available HRAM, as long as such subset uses the lower pages. An example will clarify this.

The maximum number of HRAM pages accepted by the function is 7, but typical HEPAX configurations have TWO pages (Standard HEPAX, 8k) or FOUR (Advanced HEPAX, 16k). The ROM id#'s are assigned using *the same value as the page number* – be aware that this may conflict with other ROMS currently plugged in your CL, notably **POWERCL** uses ROM id# "C", and **YFNS** uses "F" so *those two pages will have to be their id# manually re-issued to avoid any issues (!)*. You can use **ROMED** or **HEXEDIT** for that.

HEPINI will check the validity of the entry for first page, which is obviously related to the number of pages (n) chosen in the first prompt. The first page must be greater than 8 and lower than (17-n). Should that not be the case, one of the following error messages will be produced:



Even if there is a considerable amount of error protection built-in, nothing will prevent you from using this function over non-HEPRAM pages, including YFNS itself! – Therefore exercise caution as always.

Example:- Say you have used **HPX16** described above to configure 16k of HRAM, that is pages C to F contain copies of the HEPAX RAM template. You may want to use some pages for the FileSys, and others to hold other ROM images, and this done in a dynamic way.

Then the following options are available to configure the HEPAX File System with **HEPINI**:

N	PG#	Result	Comment
1	C	Page C	Can extend upwards to {C,D}, {C,D,E}, or {C,D,E,F}
1	D	Page D	Can extend upwards to {D,E}; or {D,E,F}
1	E	Page E	Can extend upwards to {E,F}
1	F	Page F	
2	C	Pages C,D	Can extend upwards to {C,D,E}; or {C,D,E,F}
2	D	Pages D,E	Can extend upwards to {D,E,F}
2	E	Pages E,F	
3	C	Pages C,D,E	Can extend upwards to {C,D,E,F}
3	D	Pages D,E,F	
4	C	Pages C,D,E,F	

Notice that the configuration can always be extended to include other pages located at **upper** addresses, but not the other way around. This is because the HEPAX code searches for the blocks sequentially to determine whether they belong to its FileSystem, starting at page 8. So once they are configured, changing the location of the first page to a lowered-number block will create a conflict.

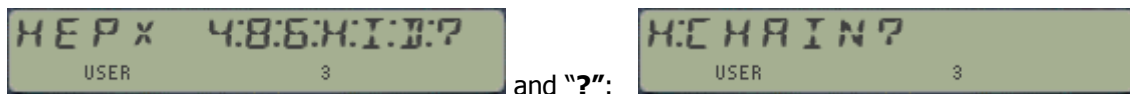
Obviously for all this to work the target pages must be mapped to sRAM – or otherwise the byte values could obviously not be changed. So it is expected that the appropriate number of HRAM pages are configured, which is the subject of the functions described in the previous section.

2.3.2 HEPAX Chain Alteration.

HEPCHN?	Recalls current CHAIN	Placed in ALPHA and Display	Sebastian Toelg
HEPCHN	Sets CHAIN	Data in ALPHA	Sebastian Toelg
RLSRAM	Releases RAM page	Pg# in X (decimal)	Sebastian Toelg

These functions provide yet additional options to configure the HEPAX chain after the first initialization, allowing for non-contiguous allocation of pages, as well as individual page removal without disrupting the complete HEPAX FileSystem. They are taken from the NEXT ROM, recently published by Sebastian Toelg (so great to see the trade is not lost!).

Note that they're located in the FAT-2 group, thus you need to use **XQ2** to execute them. For a more convenient arrangement, **HEPCHN?** is also accessible from the HEPAX main launcher, using the "?" hot-key character.



- **HEPCHN?** Returns a string to ALPHA showing the pages used by the HEPAX chain, with zeros as prefix and postfix to indicate the chain ends. The information is also shown in the display if executed in run mode. For instance, the screen below denotes pages C and D are in the HPRAM chain.



- **HEPCHN** re-configures the HEPAX chain as per the information provided in the string in ALPHA. This must always start and end with zero characters, even if there's only one page configured. Obviously all pages must be mapped as sRAM in the CL.
- **RLSRAM** releases a given RAM page (which value is in the X-register), removing it from the HEPAX chain – and closing the chain accordingly to avoid any ruptures. In RUN mode the new chain (after page removal) will be displayed for feedback information.

Comments:-

You can use **HEPCHN** to restore pages removed previously with **RLSRAM**. Executing **HEPCHN** right after **HEPCHN?** makes no modification to the HEPARAM chain.

Notice that **HEPCHN** will not take a three-zeros string as valid HPRAM chain – attempting to do so will return the error message "NULL". Note however that you could come to that situation by releasing the last page in the chain using **RLSRAM**.

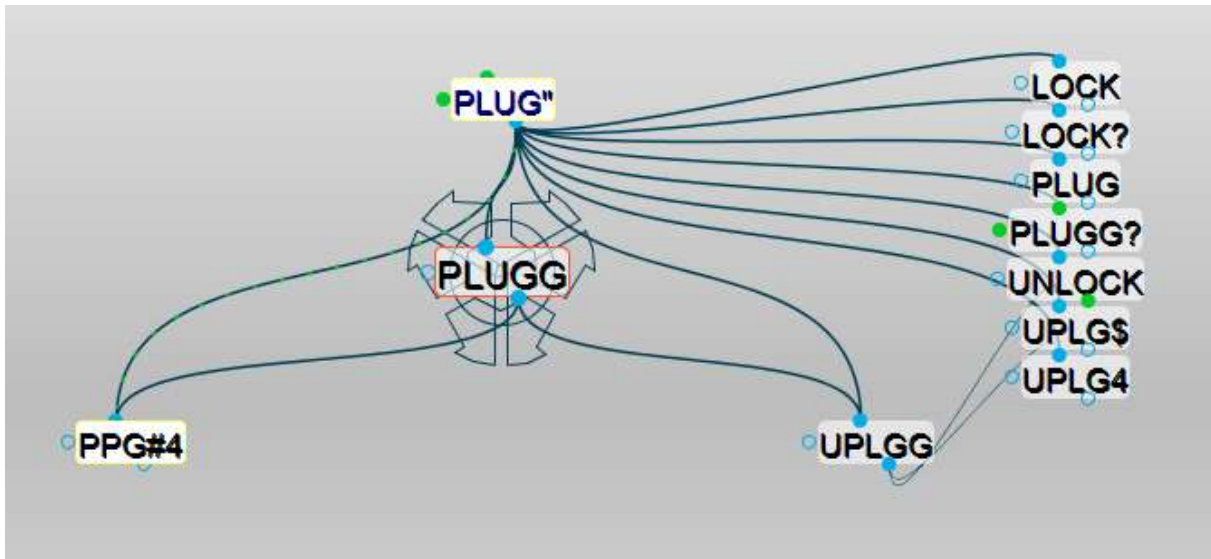
Dedicated error conditions will report the absence of a configured chain ("NO START"), a broken chain condition ("CHAIN BROKEN"), or the incorrect choices for released pages. Be careful not to remove pages or reconfigure the chain if they already contain data and are part of the FileSystem.

Note: Using the prompt **[D]** in the **HEPX** function launcher will invoke the **HEPDIR** function from the HEPAX ROM – which obviously is expected to be mapped to the MMU (or physically plugged to the CL) for this to work.

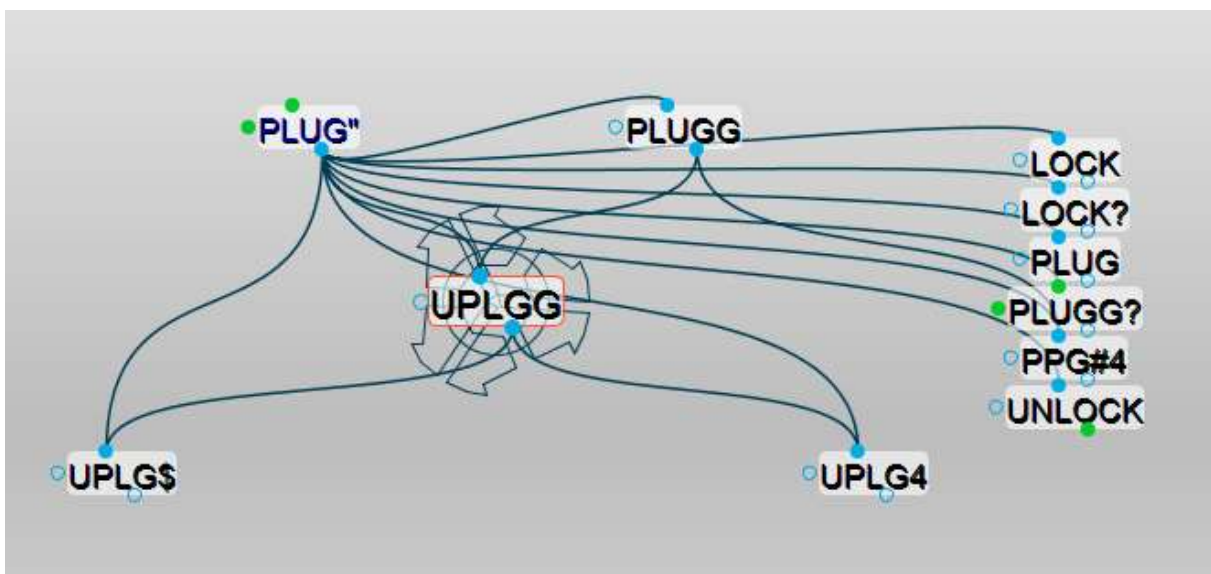
2.3.3 Page-Plug functions.

PLUGG_	PLUG page by prompt	Prompts for page: "6-F"	4k ROMS only
PLGG#4	PLUG page #4	Prompts for ROM: "F:L:S"	Take-Over ROMS
PLUGG?_	Get plugged ID#	Prompts for page: "6-F"	
UPLGG	Unplugs page	Prompts for page: "6-F"	4k ROMS only
UPLG\$	Unplugs ROM by Alpha	ROM Signature in ALPHA	

As mentioned before, the HEPAX configuration functions also take care of plugging the HEPAX ROM into the appropriate page. This is accomplished by a single function, using a parameter to define the page address. This function is **PLUGG**, or "Plug Page".



The unplugging action is performed by the reverse function, **UPLGG**. Both functions expect the page# in the X register when used in PRGM mode. In manual (RUN) mode it presents a prompt to input the destination page. As before, pressing [SHIFT] will toggle between **PLUGG** and **UPLGG** – all interconnected for quick changes and convenience sake.





Contrary to the port-based convention of the original YFNS / YFNP functions we're now referring to a page-based one, whereby the arguments of the function are the ROM id# in Alpha (same as usual) and the page# in X – removing the hard-coded dependency of the location used by the **PLUGLxx** and **PLUGUxx** functions.

The picture below (taken from the HEPAX manual) provides the relationship between ports and pages, also showing the physical addresses in the bus and those reserved for special uses (like OS, Timer, Printer, HP-IL, etc). Note that some pages (also called 4k-blocks or simply "blocks") are bank-switched. As always, a picture is worth 1,024 words:

Block Addresses			
F	F000-FFFF	Port 4, upper	
E	E000-EFFF	Port 4, lower	
D	D000-DFFF	Port 3, upper	
C	C000-CFFF	Port 3, lower	
B	B000-BFFF	Port 2, upper	
A	A000-AFFF	Port 2, lower	
9	9000-9FFF	Port 1, upper	
8	8000-8FFF	Port 1, lower	
7	7000-7FFF	HP-IL module	
6	6F00-6FFF	Printer	IR printer
5	5000-5FFF	TIME	CX system
4	4000-4FFF	Take-over ROM	
3	3000-3FFF	Unused/CX	
2	2000-2FFF	System ROM 2	
1	1000-1FFF	System ROM 1	
0	0000-0FFF	System ROM 0	
		Primary bank	Secondary bank

Note that **PLUGG** and **PLUGG?** are mutually complementary functions, as they both operate on page id# and will take or return the corresponding ROM id# from/to Alpha. You could use **PLUGG?** to interrogate the MMU about page#4, and you can use **PLUGG** to plug take-over ROMS to page#4 – by directly invoking the dedicated function **PLGG#4**, which will be covered in section 2.4 of the manual later on.

The following error conditions can occur:

- Because of dealing with pages and not full ports, **PLUGG** will only accept 4k ROMS, or otherwise "TYPE ERR" will be shown. Note however that **UPLGG** will always work, removing the MMU mapping to the selected page - so be careful not to half-remove 8k-ROMS (!)

- Main valid page# inputs are within to the 6-F range. Letters other than A-F will be inactive during the prompt, but all numeric keys will be allowed - yet values less than 6 will also be rejected, resulting in an "OS AREA" error message.
- Also a valid special input is "4", which accesses the page#4 functions – but it requires the string "OK" placed in ALPHA to accept it. Any other value will trigger a "NOT OK" error message.
- Attempting to plug a ROM to the page currently used by YFNS will also trigger an error code, to prevent accidental overwrite. The display will show "PG=YFNS" and no action will occur. If you want to relocate it you need to use one of the CL "standard" **PLUG** functions instead.
- If the string in Alpha is not a valid ROM id# you'll get "BAD ID" or "NO ENTRY" – as expected.
- If the YFNS ROM is not present (not mapped to the MMU or running on a standard 41 without the CL board) you'll get "NONEXISTENT" error.

Using Module Information for Unplugging. { UPLG\$, UPLGID }

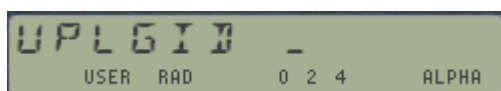
Functions **UPLG\$** and **UPLGID** introduce a new aspect to the ROM plug/unplug chapter in that it uses the text ALPHA as input, and not page or port information – which may not be known to the user at the moment of the unplugging action.

For UPLG\$ The string in Alpha is expected to be the **Page Signature**, a four-letter string written at the end of each 4k-page comprising the module, just before the checksum word at the very bottom of the page.

Typically the page signature is not known to the normal user, but power-users are of course a different kind. There is a couple of ways to find out the signature for any page:

1. Using the function **PGSIG**; which prompts for the page number in Hexadecimal format (from 0 to F), returning the signature string to ALPHA and the display. Note that it's also programmable, and than *in PRGM mode the page# is taken from the X register instead.*
2. Using **MMUCAT**; now also adding the signature to the information shown in the display. This enhances the output and provides more details as to the type of mapping for each page, which may be for a module in Flash (showing its ROM CL_ID and the page Signature), or a module in sRAM (showing the address and the Signature), or not part of the MMU (showing zeros and signature), or even a blank page with nothing plugged in – virtually or physically - (showing zeros and the "@@@@" string). All in all, a better characterization of the system configuration, *which will also be printed if the NORM/TRACE mode is set.*

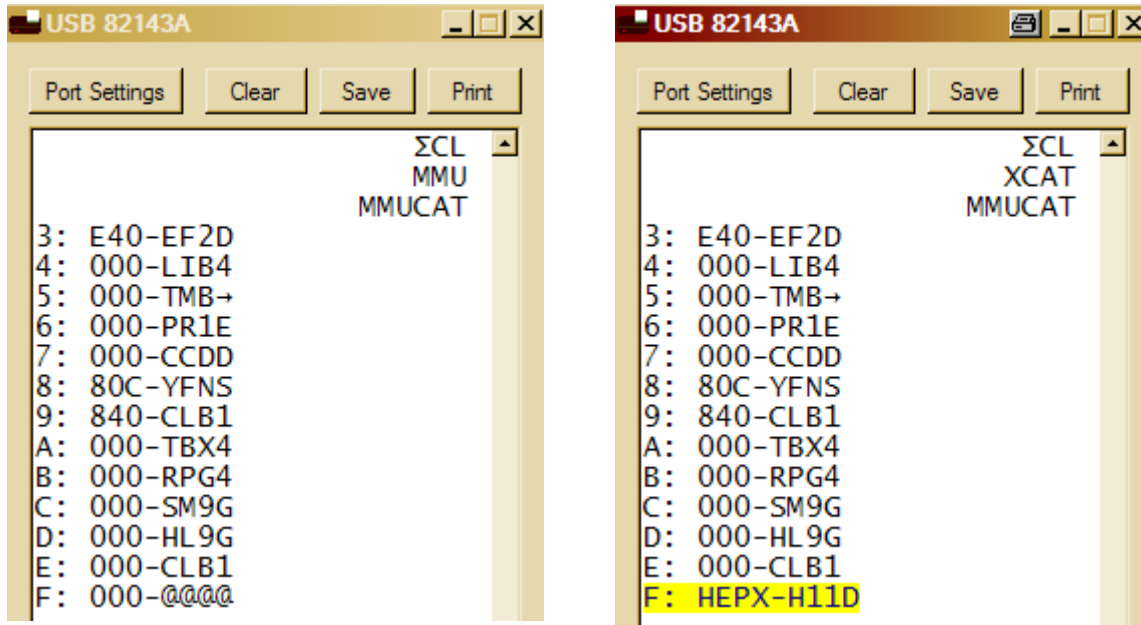
Notice that despite being a similar concept, the page signature is **not** the same as the ROM ID mnemonic used by the CL's **PLUGxx** functions – and listed by **ROMLIB** or **CLLIB**. If this is known then you can use function **UPLGID** instead, which will prompt for the mnemonic in RUN mode or take it from ALPHA during a program execution.



Note how **UPLGID** will switch ALPHA on automatically when used in RUN mode, a feature common with most other functions in the PWRCL that take alphabetical inputs.

At this point an example should clarify. The screenshot below shows the output of **MMUCAT** for a system with the USB-41 module, and a couple of MMU-plugged modules. Note that the Page#4 Library is included in the USB-41 itself, and thus the MMU address for it is zero - as it corresponds to any "real" module (i.e. not virtual).

Note also that in pages #8 and #9 the sRAM addresses are listed (as opposed to CL_ID' s) since I'm not using the Flash versions of the CLLIB or YFNS modules. Lastly, note as well the "all empty" nature of page #F in the left-side picture.



Plugging the HEPAX module from Flash to page #F changes the last line as seen on the right-side picture, showing both the CL_ID and the page signature side-by-side.

2.3.4. Security functions.

The following group of functions are a small detour, in that they aren't directly related to the CL but they come to full fruition when used on this platform.

SECURE	Activate Security	Author: Nick Harmer	Source: Data File
UNSECR	Deactivate Security	Author: Angel Martin	
XPASS	Change Password	Author: Nick Harmer	Source: Data File

Here we have a nice practical application of advanced system control. Use these functions to manage a password-protection scheme for your CL – so nobody without authorized access can use it.

They were published in Data File back in 1987 by Nick Harmer, and implemented in Q-RAM devices (a.k.a MLDL). Obvious caveat there was that removing the MLDL from the machine dismantled the whole scheme – but the CL has made it possible as integral part of the core system now.

The protection works as follows:-

1. Function **SECURE** activates the security by setting the protection flag. The execution also switches off the machine. This sets up a process executed on each CALC_ON event, causing to prompt the user for the password during the start-up process.
2. Function **UNSECR** deactivates the security by clearing the protection flag.
3. Function **XPASS** allows the user to change the password from the default one to his/her favorite one. The length of the password is limited to six (6) characters.



Enter code (up to 6 chrs. long) and end with **[R/S]**

Inputting the password is very simple but very unforgiving as well: at the prompt "PASSWORD=?" just type the letters one by one until completing the word, and you're done. If you make a mistake the machine will switch itself off and it'll be "groundhog day" all over gain – until you get it right.

Each keystroke will be acknowledged by a short tone, but no change to the display – so nothing like "*****" as you type the word. If the wrong letter is entered a lower-pitch sound will be heard and the calculator will go to sleep.

Be especially careful when entering a new password code – as there is no repeat input to confirm the entry, so whatever key combination you type will be taken when ending the sequence with R/S. The initial password ("factory default", so to speak) is "CACA".



Enter code (up to 6 chrs. long) and end with **[R/S]**

Here again it comes without saying that this will only work when the PWRCL_Ext module is mapped to a sRAM block in the MMU – otherwise none of the ROM writing will work.

Note: this is how you'd get yourself out of trouble if somehow you forgot the right code: call **YRES** (or do a memory lost) to disable the MMU, and then reload the POWERCL from flash – which has the protection flag cleared. Map it to the right page and enable the MMU again – you're back in charge.

2.3.5. Encrypting and Decrypting RAM contents.

YFOG	Scrambles RAM	See description below/	*WARNING*
YCRYPT	Driver for FOG	Follow program prompts	*WARNING*

If you're concerned about the security of the data and programs held in your RAM pages here is the ultimate encryption facility to completely cover your tracks and protect the system to the paranoia stage.

YFOG will scramble the RAM contents using a 6-characters long encryption key provided in ALPHA, starting from the address in the S&X field of X (thus a NNN is expected), and until the bottom of that page. Repeating the operation with the same encryption code will restore the contents to its original state, so the operation is reversible – as long as you remember the key used to encrypt it in the first place.

YCRYPT is a nice and easy driver program for **YFOG** that takes care of preparing the required inputs for you. No additional "precautions" are added, so the "ADR _ _ _ _" input will accept any HEX characters and not only valid addresses.

As these functions only operate in RAM the OS area is safe, even if you attempt to encrypt it. Ditto for every page plugged to a module in Flash memory – but watch out for the RAM-plugged pages (like HEPAX RAM, or any other module you have residing in sRAM). That's why the confirmation message "OK? Y/N" will also be prompted when calling this function – even in a program.

Needless to say things can get hairy pretty quickly if you mess with critical areas, like the polling points. **YFOG** will not modify the contents of locations 0xpFF4 and above within the page, but that doesn't guarantee a trouble-free result – because if polling points are active who knows what will be there where they're pointing at AFTER the encryption!

1	LBL "YCRYPT"
2	-SYS EXT
3	RCL Q
4	"PWD="
5	PMTA
6	ASTO L
7	ALENG
8	6
9	X#Y?
10	GTO 01
11	RCL Z
12	FOG
13	TONE 7
14	LBL 02
15	CLA
16	ARCL L
17	GTEND
18	LBL 01
19	TONE 0
20	"#PWD<>6"
21	AVIEW
22	GTO 02
23	END

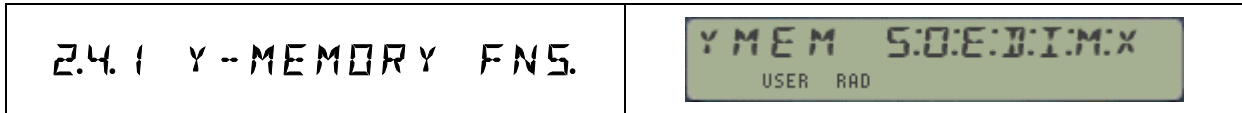
YCRYPT is a trivial-looking program that just manages to prepare things for **YFOG**. The listing is shown on the right, note the usage of the undocumented function "**-SYS EXT**" that prompts for the target address and places it in the S&X field of status register Q(9).

YCRYPT uses a couple of functions from the AMCOS/X module, which obviously should also be plugged in the MMU.

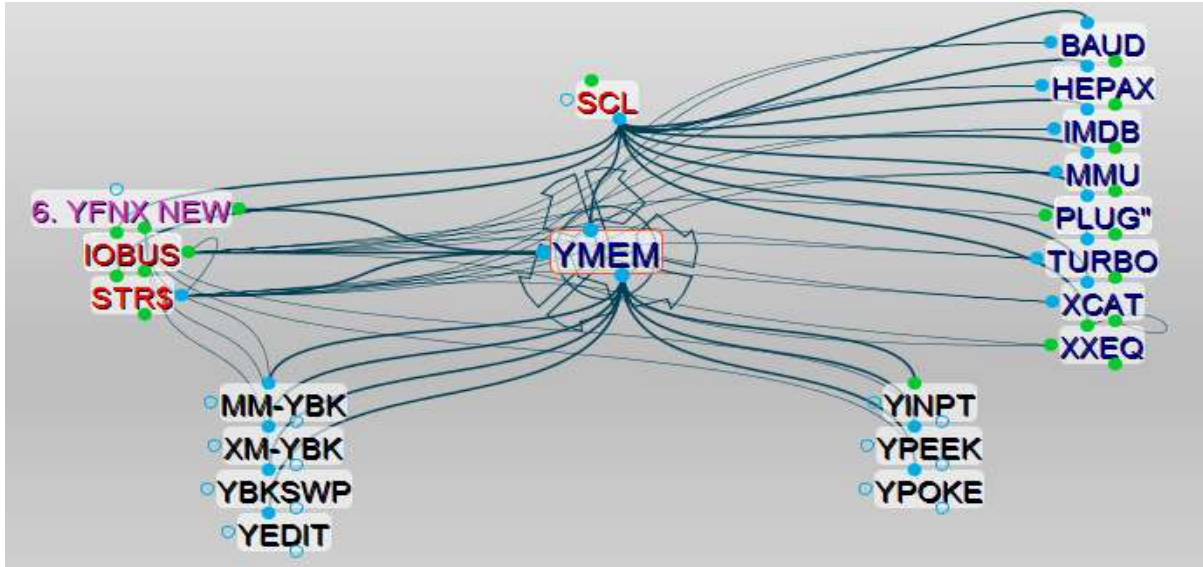
GTEND is also available in FAT1, so you could use **XQ1 "GTEND"** instead. Or even just stop there (STOP, RTN) but it's safer to leave the scene after the job is done (or the error encountered).

A piece of trivia: **YFOG** must be located in the main FAT in order to be callable from within the **YCRYPT** code: this is a limitation that won't affect your programs located in RAM – or even located in other plug-in modules - as long as they are not plugged in the same PORT as the POWER_CL.

The original **FOG** function was written by Derek Amos, and published in PPCCJ, V12N5 p3.



2.4.1. Alternate Blocks and Expanded memory



Welcome to the most recent addition to the PWRCL_EXT module. These functions are not included in the POWER_CL version, and have only recently been made available. They require the YFNX module to operate. The following error message will be displayed if the FNX module is not found:



Note that this check is based on the FAT entry order for function **MMU?**. This is less fool-proof than looking into the module checksum information, but it's more flexible approach that will work with different revisions of YFNX as long as such function order has not changed.

The intent here has been to include in the launcher the most frequently accessed functions, like **YPEEK** and **YPOKE**, together with a couple of favorite ones: **YEDIT** and **YINPT** - described somewhere else in this manual.

YEDIT	sRAM mini-Editor	Input ADR, then Y/N	WARNING
YPEEK	Peek Word Value	Location syntax in ALPHA	
YPOKE	Poke Word Value	Location/Value in ALPHA	WARNING
YINPT	Universal Y-input	Prompts for characters	WARNING
MM-YBK	Main Memory to Block	Destination Block# in X	Includes Status regs.
ST-YBK	Status Regs. To Block	Destination Block# in X	
XM-YBK	Extended Mem. to Block	Destination Block# in X	
YBK-MM	Block to Main Memory	Source Block# in X	Includes Status Regs
YBK-ST	Block to Status Regs.	Source Block# in X	
YBK-XM	Block to Extended Mem.	Source Block# in X	
STYSWP	Status Regs Exchange	Swap Block# in X	
YBKSWP	Alternate blocks swap	Source in X, Dest. in Prompt	
MMYSWP	Main Mem Exchange	Swap Block# in X	Includes status regs.
XMYSWP	Ext-Mem Exchange	Swap Block# in X	
YRESET	Resets (Disables) the MMU	Use it when YFNS is "lost"	
?YFNX	Tests for YFNX presence	Shows error if not	(doesn't SKIP line!)

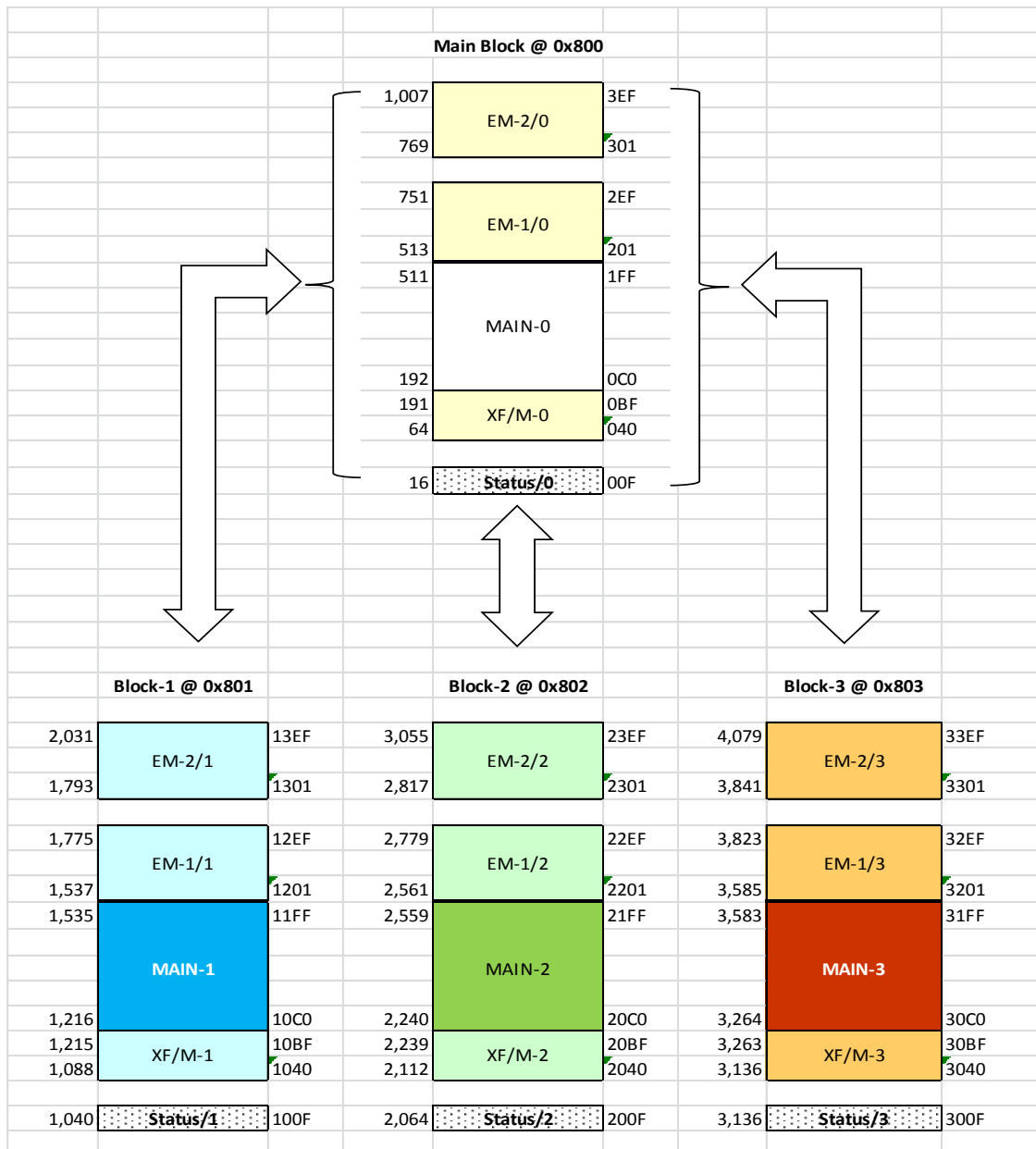
The CL Expanded Memory Sets

The CL board has four blocks in sRAM that can be allocated for the 41 system RAM. Of these, only the first one (0x800) is utilized by the OS, holding the registers 0-3FF – that is the complete calculator memory set, including both Main and all Extended Memory.

It is therefore possible to use the other three blocks to hold backup copies of the first (default) one, or alternate sets of memory and programs. This requires a few utility functions to store, recall and exchange (swap) the block contents – which is the main subject of this section.

The operation can be done at the complete block level, or selectively for Main and Extended memory only. Note that the Main memory transfers always include the status registers as well.

All functions require the alternate block# to be used in the corresponding action. The only valid inputs are 0, 1, 2, or 3 – denoting the blocks at addresses 0x800, 0x801, 0x802, and 0x803 respectively. The picture below should graphically depict the CL Expanded Memory structure and data transfer possibilities:



To prevent accidental data loss, the operation requires the confirmation string "OK" in ALPHA to proceed if the destination block is zero (i.e. the main RAM used by the OS).

These functions are fully programmable, but you should be careful about altering the main memory (or status registers) during a program; as you may be overwriting the program itself, or other OS parameters like the program pointer or line number. This will likely result in a MEMORY LOST event.

A look under the Hood: General-Purpose Functions.

The function set provides choices for the particular section to back-up or restore (Main & Extended memory, Status registers), so you don't have to worry about physical memory boundaries or type of transfer. Besides those, two general-purpose functions **YMCOPY** and **YMEXCH** (not to be confused with **YMOVE** and **YSWAP**, which operate on the MMU settings instead) are also available that allow a *flexible selection of memory sectors to be transferred*, defining a sector as a 256-register set within the block. The input parameters are expected to be in a control NNN stored in the X-register, as follows:

Source and Destination:

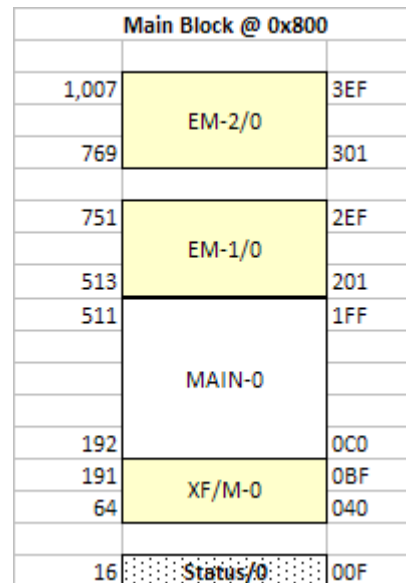
input X<13> = source
 input X<12>= destination
 (Valid values are 0/1/2/3)

Data fields:

input X<3:0> = blocks to transfer,
 one digit (nibble) per 256-word block
 (16 sectors for complete block)

The table below shows the values used on the standard functions internally, where "b" denotes the alternate block# (1,2, or 3) used for the backup or restore:

Function	Swap Flag	Control NNN
MM-YBK	Clear	0b-00000000-00F9
MMYSWP	Set	0b-00000000-00F9
ST-YBK	Clear	0b-00000000-0001
STYSWP	Set	0b-00000000-0001
XM-YBK	Cleat	0b-00000000-FF06
XMYSWP	Set	0b-00000000-FF06
YBK-MM	Clear	b0-00000000-00F9
YBK-ST	Clear	b0-00000000-0001
YBK-XM	Clear	b0-00000000-FF06



The diagram on the right shows the beginning and ending addresses for each of the significant sections within the main block; in decimal to the left and hex to the right. With exception of the general-purpose functions **YMCOPY** and **YMEXCH**, the appropriate addresses are chosen automatically by each of the memory transfer functions, all completely transparent to the user.

Lastly, to prepare the control NNN to use with the general-purpose versions of the memory transfer functions, you can use any of the HEX Entry functions available in the PWRCL_EXT, such as **CDE**, **HEXKB**, **HEXIN**, as well as the powerhouse memory editor **RAMED**. See their description in later sections of this manual.

Inputting Y-Control strings with YINPT.

The reason why characters "<->" and ">" are so relevant is the formatting syntax required by many of the functions, like **YPEEK**, **YPOKE**, **PLUGxx**, etc. To that effect a most useful function within this group is **YINPT**, which redefines the keyboard as a hex entry {0-9, A-F}, plus a few special control characters, as follows:

- [J] adds the control character ">" to the display
- [Q] adds the control character "<->" to the display
- [K] adds the string "16K" to the display – for bank-switched modules
- [L] adds the string "DBL" to the display – for 8k modules
- [M] adds the string "RAM" to the display – for single 4k modules
- [SST] adds the string "MAX" to the display – for 16k modules
- [ENTER ^] adds three zeroes to the display
- [<-] BA removes the last character (or groups above), or cancel out if Empty
- [R/S] terminates the entry process, and copies the display content to ALPHA.

Using this function expedites the construction of the Alpha strings required by all other Y-Functions, make sure you have it assigned to a handy key as it's likely to be used quite frequently – or alternatively use it directly from the **YMEM** launcher, option "I".

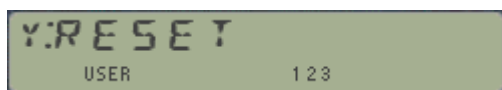
When used in PROGRAM mode, the prompt "Y:" will be replaced by the text in ALPHA – so you can have your custom prompts.

Notice that because it uses the display as repository (ALPHA is not altered until the end), the valid length accepted by **YINPT** *is limited to 11 characters* (not counting the prompt). This is sufficient for all parameter input purposes within YFNS functions, but remember: if you exceed this limit characters will be "lost" from the left. **YINPT** is used in other FOCAL programs in the PowerCL module, such as **YEDIT** and **DLD48**.

Resetting the MMU.

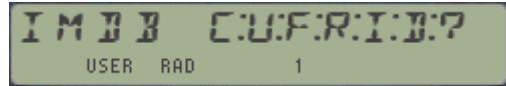
- **YFNZ?** is completely equivalent to **YFNS?**, in fact is just a code stub that invokes the latter. It must be in the POWERCL module for subroutine purposes. Incidentally, this is how **PLUGGX** checks for YFNS being currently mapped to the target page, and discards the request if so.

[Note: there's an ulterior motive for having this function included in the POWERCL module, and such is as a back-door recovery method from a "lost YFNS" contingency. This may occur when the YFNS is overwritten with another module image by the MMU, typically as result of inadvertently using **PLUGxx** on the same location. If that's the case, **YFNZ?** will automatically disable the MMU so you can take charge again and rebuild the MMU to correct the issue - with no need to resort to ML or removing the batteries anymore. Upon completion the following message is displayed:

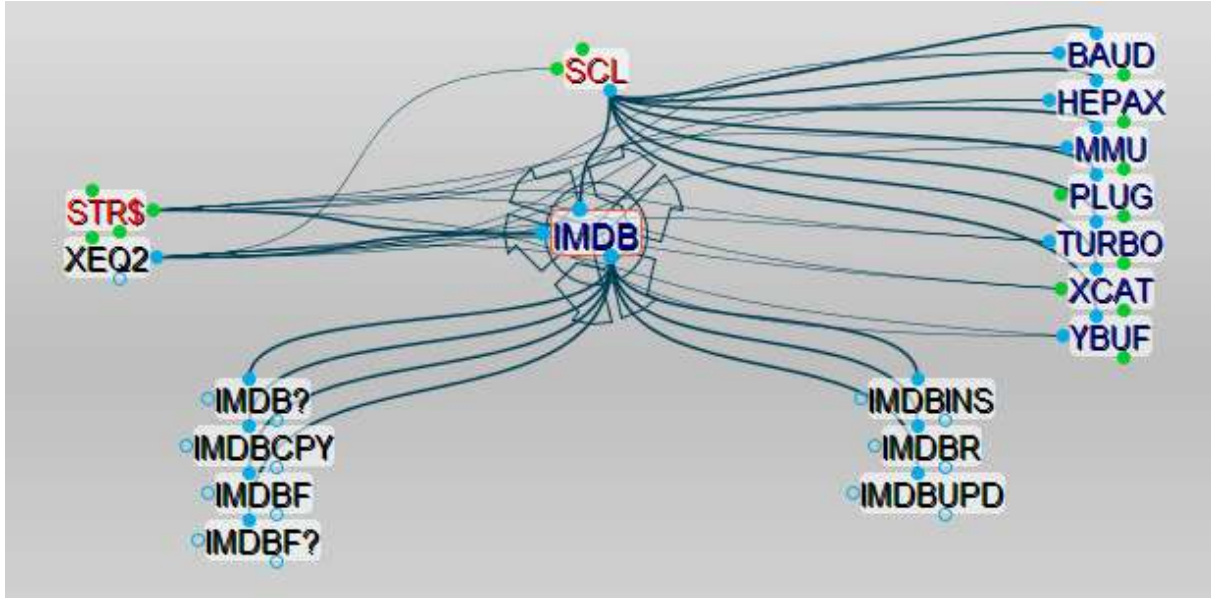


- **YRESET** was added in revision "J". It provides a more purposed approach to the same task, i.e. looking for the YFNX module and effectively disabling the MMU (resetting it) if it's not found - albeit only if the "OK/OKALL" string is in ALPHA – to avoid accidental disabling.
- Lastly **?MMU** is the same as MMU? In the YFNS module - plus also writing either 1/0 in X depending on the result. The stack contents are lifted

2.4.2 IMAGE DATABASE



2.4.2. The new kid on the block.



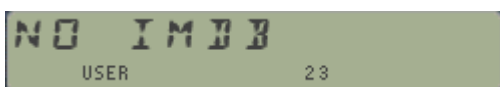
One of the additions to the initial CL's bag of tricks was a set of functions to manage the ROM Image database. It's not in the scope of this manual to describe them, you'd have to read the CL manual for that; but this launcher is available to group all of them under a single function in the POWERCL. **Be very careful** not to use the options by mistake – correcting errors in the IMDB is time consuming!

IMDBCPY	Copy IMDB to RAM	Makes a copy in 0x805	WARNING
IMDBUPD	Update IMDB from RAM	Flashes the entire sector!	WARNING
IMDBF	Activates IMDB in Flash	Uses copy in Flash	
IMDBR	Activated IMDB in RAM	Uses copy in sRAM	
IMDBINS	Inserts new IMDB entry	Data in ALPHA	WARNING
IMDB?	IMDB Entry Data	Data in Alpha	
IMDBF?	Checks if IMDB in Flash	Yes/No	

The IMDB is the only reference used by all **PLUGxx** and **UPLUGxx** function, be that the "native" port-related ones in the YFNS, or their page-related counterparts in the POWERCL. It's therefore not to be confused with the internal database within the POWERCL, used by functions **ADRID**, **CLLIB** and **ROMLIB** – which purely perform an informational role for enhanced usability.

IMDB? has similarities with **ADRID** in that both retrieve details on the ROM specified in the argument. For **ADRID** such is its flash or RAM address (returning the ROM ID in Alpha), whereas **IMDB?** uses the string in ALPHA – either ROM ID or address are valid inputs.

Obviously the CL is a must requirement; so don't try this on your favorite emulator or "plain" machine, where you'll get the friendly reminder (courtesy of the YFNX module):



Differences between the IMDB and the PowerCL ROM Library.

The PowerCL has its own image Library included in bank-2. This local library (currently occupying about 1,350 bytes) is the source of information for functions **CLLIB** and **ROMLIB**, covered earlier in the manual. The entries are arranged alphabetically, and they have field descriptors for the module size and type – used by those functions, as you already know.

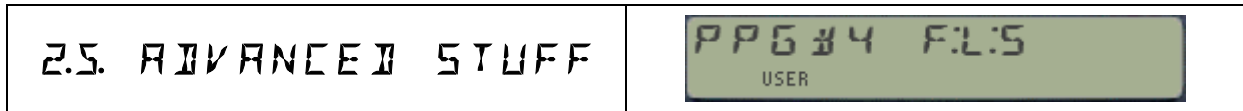
There are 223 entries in the ROM table. Each entry takes up 6 bytes as follows: 4 for the mnemonic, one for the address in flash, and the last one for the size/type information combined. Besides, a table header and footer (of 6 bytes each) are also needed for the search algorithms.

000	<Table Boundary>	
000	<Table Boundary>	
000	<Table Boundary>	
000	<Table Boundary>	
000	<Table Boundary>	
000	<Table Boundary>	
001	"A"	<i>Advantage Pac</i>
034	"4"	<i>HP</i>
031	"1"	<i>12k</i>
210	"P"	
20D	size / type	<i>Size in XS digit</i>
008	address	
001	"A"	<i>Advantage Applications</i>
001	"A"	<i>J-F Garnier</i>
004	"D"	<i>4k</i>
216	"V"	
10D	size / type	
08D	address	

Arguably there is redundancy in maintaining a duplicate of the IMDB, and a potential source of errors if there is divergence between both. In the ideal scenario this should not be required; however it was coded very early in the days of the beta version of the CL board, before the IMDB was implemented. It is also an integral part of functions **ADRID**, **MMUCAT**, and **PLUGG?** – which makes it rather difficult to eliminate now after its pivotal character.

An obvious limitation is that the PowerCL knows nothing about new IMDB entries made after its release, or those you may have added to it in your own CL system.

It will also get out of sync if you re-locate ROM images within the IMDB – although this is quite a corner case, not likely to occur with the 99.99% of users.



2.5.1. Using Page#4

As mentioned previously page#4 is a special case that requires its own dedicated (un)plugging functions, not covered by **PLUGG** or the native (**U**)**PLUG** ones either.

PPG#4	Plugs ROM in page#4	Prompts F:L:S	WARNING
UPGG4	Unplugs ROM from p4		

The 41 OS reserves Page #4 as a special location. There are frequent checks done during strategic moments to specific locations that can be used to take control on the system, even over the OS itself if that was required – as it happens with the diagnostics executed from the different SERVICE ROMS.

Because of that, only "take-over" ROMS can be plugged in page#4. They have been written specifically for it and will either take complete control of the system (like the FORTH and service modules), or drive it from their own directive (like the LAITRAM module).

Function **PPG#4** prompts for the ROM to plug into the page, options being just those three mentioned above: FORTH, LAITRAM, or SERVICE modules – by their initials: "**F:L:S**". Once the selection is made the function transfers the execution to a hidden FOCAL program that writes the appropriate entries into the MMU registers, so that the mapping is correct. Refer to the CL manual for details on this.

WARNING: Be aware that once the order is complete you'll be at the mercy of the plugged module. Going back to the "normal" OS may not be as simple as you think, especially with the Service ROM plugged – which requires removing the batteries, then clearing the MMU entry with the MMU disabled after you switch it back on.

For the other instances it is possible to "exit" back to the OS, and thus you could execute **UPGG4** to unplug the module from the page. Obviously no inputs are needed in this case.

Note that because of the name not directly key-able using **XEQ** (an intentional measure) you'll have to use another approach to invoke **PPG#4**. It's a trivial task with the CCD-style CAT'2, either during the catalog run or through a previous assignment to any USER key. Of course as a CL owner you're only one **YPOKE** away from a permanent solution if POWERCL resides in sRAM ☺.

And what about the Library#4?

The Library#4 is not one of the choices offered by **PPG#4** because *it's assumed to be already installed for the POWERCL module to work*. This configuration must be done manually. This is an important point, so make sure to always have the Library#4 plugged in order to use the POWER_CL module. The library contains routines used by many functions in the module, thus it must be present for proper execution. Failure to do so will create unexpected issues ranging from unpleasant to harmful events!

WARNING: Be aware that just plugging/unplugging the Library#4 (once you have it burned in Flash or copied in sRAM) will not check for its presence. It is therefore strongly recommended to power-cycle the machine in order to perform the Library#4 existence check. As always, refer to the Library#4 documentation for additional details.

2.5.2. RAM and ROM Editors

Placing the ROM image library on bank 2 freed up a large amount of space for additional functions to be included in the POWER-CL module, as can be seen by looking at its full-FAT list. The choice of functions added over previous CL_UTILS incarnations was clearly meant to have a comprehensive and self-contained function set that included some the best examples ever written for the HP-41 system. RAM and ROM editors are no doubt amongst these, and as such are available in the POWER-CL.

RAMED	RAM Editor	Uses GETKEY [KEYFNC]	WARNING
ROMED	ROM Editor	Uses Partial Data Entry	WARNING
YEDIT	CL sRAM Editor	Uses AMC_OS/X Module	

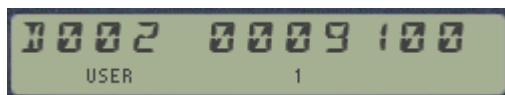
Editing RAM memory with RAMED.

Written by Håkan Thörngren, this powerful RAM editor rivals with (and exceeds it in several aspects) the ZENROM implementation. It was first published in PPCJ V13 N4 p26-, you're encouraged to check his original contribution for a complete description of the functionality and usage.

The starting address is taken from the X register in RUN mode (as decimal value between 0 and 999), or from the program pointer in PRGM mode. The display shows two distinct fields, with the nybble & byte section shown on the left side and the actual register content shown on the right – as a 7-digit scrollable field controlled by the USER and PRGM keys – very much like the CX's ASCII file editor **ED**.

Nybble D (the 13th within the register) is selected upon start-up, with the cursor centered in the middle of the field and its value blinking on the display. At this point you can use the control characters to move between both areas and within the fields, or the digit keys plus A-F to input the nybble HEX values being edited. Scrolling includes a tone to signal the wrap-around condition within the register, as the nybble being edited is updated in the address field on the left. Without a doubt, a real tour-de-force and a masterful implementation.

The screens below show a couple of examples, editing the leftmost nybble of the Y register (address: D002) and the rightmost digit of the X register (address 0003). The screenshots don't capture its magic; you really need to use it to appreciate its simple and powerful functionality.



The control keys for RAMED are as follows:

- [USER]: moves down to the previous nybble or position within the field
- [PRGM]: moves up to the next nybble or position within the field
- [+]: moves up to the next register
- [-]: moves down to the previous register
- [.]: the Radix key moves between both fields, used to change the register address
- [1]-[9],[A]-[F] the nybble value being edited
- [<-] back-arrow cancels out and exits the editing
- [ON]: turns the calculator OFF

RAMED is completely located in bank-2, with only the function name and a small code snippet in the first bank to transfer the execution. I have only minimally altered the source code to take advantage of the CX- and Library#4 routines.

RAMED uses a key-detection technique more power-demanding than the Partial Key Sequence, thus will drain on the battery life if used extensively. Do not leave it run idle for a prolonged time.

Editing ROM areas with ROMED.

Written by W. Doug Wilder, another MCODE master - this ROM editor has all the basic functionality required for the most common needs; perhaps just a couple of notches below the tremendous HEPAX's **HEXEDIT** – but in a much more concise foot-print implementation and not exempt of wonders on its own.

The initial prompt requests the address to edit, ranging from 0x0000 to 0xFFFF as you would expect. Once entered, the display is identical to that one in **HEXEDIT**, with three distinct fields showing the address being edited, the current value, and three underscore characters where the new value will be written as the input progresses.

Usual rules of the game apply: the first character can only be 0,1,2,3; and obviously there must be a MLDL-RAM block for the input to be actually written in. A nice touch (lacking in **HEXEDIT**) is a "ROM" message shown when the destination is read-only.

The control keys for ROMEDIT are as follows:

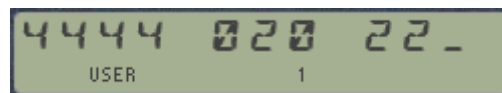
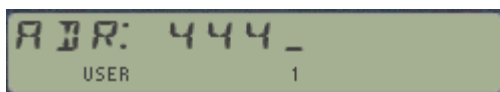
[**SST**] : moves up one word
[**BST**], [**TAN**] moves down one word
[**1**]-[**9**],[**A**]-[**F**] the nybble value being edited
[**ENTER**^]: inputs three zeroes as word value
[<-]: first back-arrow prompts for a new address, second exits the editing
[**ON**] turns the calculator OFF

ROMED is completely located in bank-1. Besides the changes "*of rigueur*" to use the Library#4 routines, I have made a couple of enhancements to the original implementation, adding the underscores for the edited field and the [**ENTER**^] control key for a closer resemblance to the HEPAX implementation in **HEXEDIT**.

ROMED can only edit the first bank, so the only missing functionality is perhaps this; no access to the other banks in bank-switched modules (like the HEPAX, Advantage, Timer, or POWER_CL itself). Certainly not a big deal in the 90% of the cases; and sure enough well worth the admission price.

[**Note:** The other missing functionality is the ability to do live-editing of polling points and other OS-controlled hot addresses, which the HEPAX manages by preventing the calculator from going into light sleep – definitely hardly used in the 99,9% of the cases.]

The screenshots below show editing of the Library#4 contents on-the-fly – a real godsend for MCODERS to quickly test small code changes without having to re-compile / rebuild the ROM images.



ROMED uses the partial-key entry technique, more gentle on the battery drain requirements – and incidentally also the reason why it cannot be located on bank-2, as a technical detail.

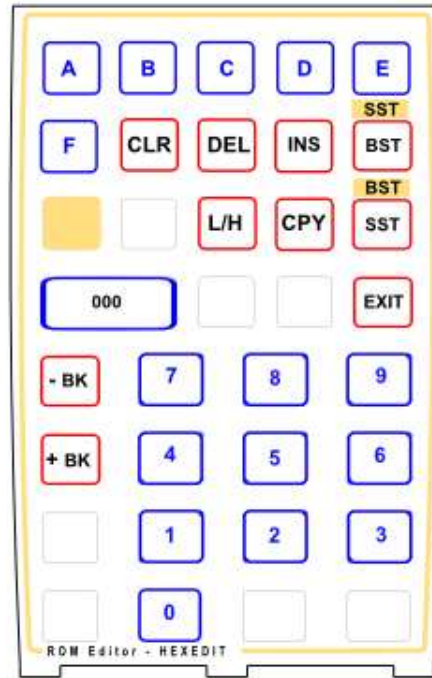
Final remark.- The original **ROMED** is available in the DISASM module, under the name **WROM**, and **RAMED** is also included in the RAMPAGE module, named **RAMEDIT** there. Thanks to its gigantic module image library there's certainly no shortage of modules to use on the CL, but having all these functions included in a lean 4k-footprint ROM is very advantageous from the usability and compatibility standpoints, eliminating the need to alter the system configuration to access another module containing the desired function.

RAMED and ROMED Overlays.

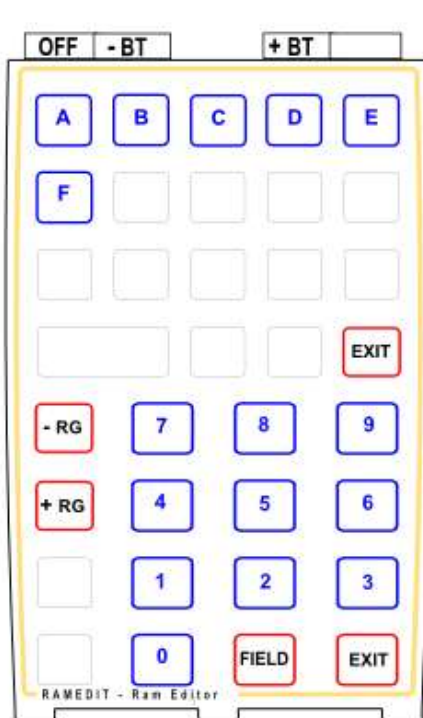
The figures below show two possible overlays for the Ram and ROM Editor functions, **RAMED** and **ROMED**. For comparison purposes, I have also prepared similar virtual-overlays for the more capable versions of the same functions included in the ZENROM and the HEPAX. Interestingly, the overlay for **ROMED** could be simply a subset of the **HEXEDIT** overlay – so it'd appear killing two birds with the same stone.



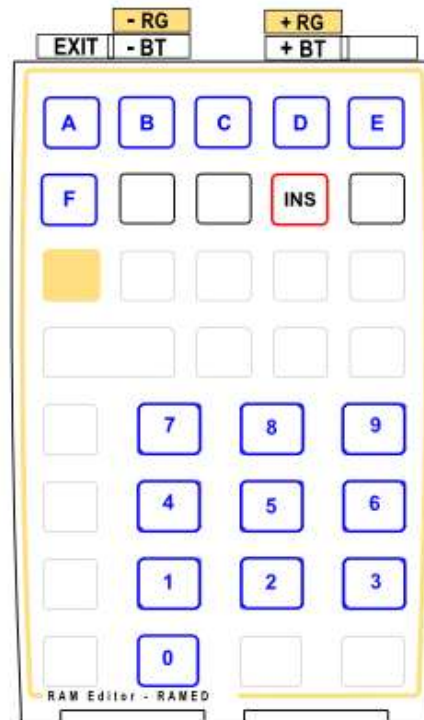
ROMED - Wrom



HEXEDIT - Hepax



RAMEDIT - GetKey



RAMED - Zenrom

2.5.3.- YEDIT- Quick & Dirty sRAM Editors.

RAMED and **ROMED** are capable of editing the complete content of the RAM and ROM plugged to the CL via the MMU settings, but obviously cannot be used to edit the content of the sRAM on the CL board. **YEDIT** closes that gap, in a minimalist approach that provides a simple but elegant way to drive **YPEEK** and **YPOKE** instructions automatically, so that the sRAM words can be edited.

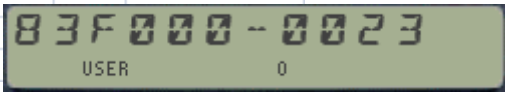
The execution starts by asking for the starting sRAM or Flash address, as a six-digit hex format. It then gets into a loop reviewing every word in a sequential manner, using [R/S] or [Y] for each consecutive word. Use [N] to get into the editing screen, input the new values (4-digits are required) and follow with [R/S]. It comes without saying that **YEDIT** isn't able to edit the data in Flash sectors, although *you can use it to review its contents*.

Note that some functions from the AMC_OS/X module are used, thus you'll need to have it plugged as well – one more reason to **always** have it on-line.

Listed below are a couple of small FOCAL programs that illustrate the functionality present in **YEDIT**. The first one only reviews the contents (useful to check whether the block contains the expected information), and the second also allows actual changes to it. Like the final program in the POWERCL module, these example programs also use functions **WSIZE** and **ARCLH**, from the AMC_OS/X module. The final version of **YEDIT** also uses **PMTA**, instead of **APPEND**.

The simple version on the right allows moving to a different address without re-starting the program, simply add (or subtract) the offset from the current one, entering it in X (in decimal) then pressing [+] (or [-]) just before **R/S**. For **YEDIT+** on the left you'll need to re-start the program and input the new address.

01*	LBL "YEDIT+"		01*	LBL "YEDIT"
02	YINPT		02	YINPT
03	ASTO X		03	ASTO X
04	12		04	12
05	WSIZE		05	WSIZE
06	CLX		06	CLX
07	LBL 05 ←		07	LBL 05 ←
08	CLA		08	CLA
09	ARCL Y		09	ARCL Y
10	ARCLH		10	ARCLH
11	" - -8888"		11	" - -8888"
12	YPEEK		12	YPEEK
13	AVIEW		13	PROMPT
14	Y/N?		14	E
15	X<0?		15	+
16	GTO 02		16	GTO 05 ←
17	E		17	END
18	+			
19	GTO 05			
20	LBL 02 ←			
21	YCL-			
22	APPEND			
23	YPOKE			
24	GTO 05			
25	END			



The screenshot above shows the data at address 0x83F000, first word in the last sRAM block in the V2 CL systems – my customary location for the Page#4 Library.

2.5.4. Reallocating MMU entries.

YMOVE	Moved MMU mapping	FROM: in Y; TO: in X	*WARNING*
YSWAP	Swaps MMU mappings	FROM: in Y; TO: in X	*WARNING*

It is often the case that you'd like to have the MMU mappings changed, to either move a current entry to a different page (freeing the original one), or even to exchange two entries. Granted this is doable using the standard **(U)PLUGxx** Y-functions, but such an approach requires knowing the module ID's, and/or the sRAM addresses where they are physically located.

New additions to the POWERCL allow doing the changes *just using their logical addresses*, i.e. the page numbers the MMU entries point to. This is much faster and simpler, yet also provides nice ways to get you in trouble if not used carefully.

In revision "K" both **YMOVE** and **YSWAP** are proper MCODE functions. Because they are in the PowerCL, the page where the PowerCL is located will be excluded – or otherwise the program pointer would "get lost" in the middle of the execution – sort of removing the floor from their feet while they're in operation. Attempting to do so will show the "NO SELF" error message.

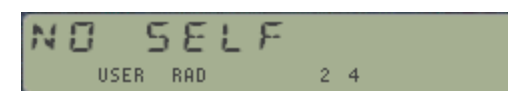
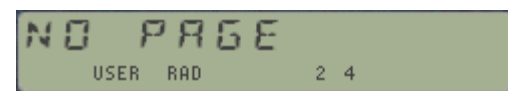
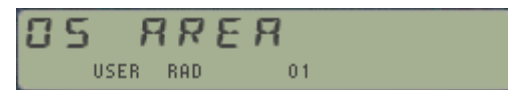
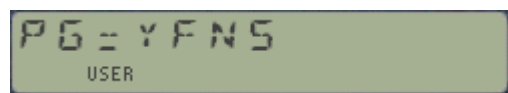
Obviously the OS pages must be avoided, as well as those system-reserved like the printer or HP-IL. Finally, revision "L" added protection to the YFNS page in a MOVE action as well. Try it!

These functions are meant to be used with 4k-ROMS, since only the pages in the TO: and FROM: inputs will get reallocated - regardless of which other logical connections may exist. And you'll be glad to know that they *support bank-switched modules* (i.e. HEPAX, OS/X3).

01	LBL "MMUSWP"	
02	,	
03	WSIZE	
04	RDN	
05	XEQ 00	<i>read the TO: page</i>
06	ASTO T	<i>save in T</i>
07	XEQ 00	<i>read the FROM: page</i>
08	ASTO L	<i>save in scratch</i>
09	XEQ 02	<i>write FROM: into TO:</i>
10	R^	
11	STO L	<i>save in scratch</i>
12	RDN	<i>write TO: into FROM:</i>
13	LBL 02	
14	"804"	
15	ARCLH	
16	" -0"	
17	ARCL L	
18	YPOKE	
19	X<>Y	
20	RTN	
21	LBL 00	
22	"804"	
23	ARCLH	
24	" -0-1111"	
25	YPEEK	
26	ASHF	
27	X<>Y	
28	END	

The FOCAL program listed on the left shows an alternative unrestricted version for **YSWAP** – use it judiciously.

Besides the restriction not to use it with the PowerCL itself, other obvious limitations for the FOCAL program (but not for the MCODE function!) include not moving the YFNS and AMC_OS/X modules either. It's only valid for non-bank switched modules



2.5.5. Calculator Flash Backup & Restore.

YWALL	Backs up to Flash	"OK" or "OKALL" in Alpha	*WARNING*
YRALL	Restore from Flash	"OK" or "OKALL" in Alpha	*WARNING*

The MMU content is preserved during a MEMORY LOST event, and the same is true with the sRAM on the CL board. So using RAM for a complete calculator backup and restore is not a bad idea at all, and it will allow you different setups or complete configurations to be swapped back and forth directly from sRAM.

However sRAM contents will be lost if the batteries are removed from the calculator for a certain period of time – longer than what it takes to reset a small glitch, but shorter than it used to be for the standard 41, - due to the increased power demands from the CL board.

Early CL beta user Geoff Quickfall prepared a few FOCAL programs to commit the calculator contents to FLASH, so that *even without the batteries* it'll be preserved for a restore at any later time. It's a powerful concept, but it doesn't come free from pitfalls if you're not careful.

- The first consideration is related to the Flash write function and you should read and understand all about it in the CL manual. Specifically pay strong attention to the recommendations about the fresh **battery state** before performing any flash-write operation.
- The second one is that **YWALL** will pick certain hard-coded FLASH locations as destination for the backup, **so both 4k blocks 0x1FE and 0x1FF will be ERASED** by **YFERASE**. Note that earlier versions of CLUTILS used sector 0x0D8-0x0DF instead. This was moved to the current location to avoid erasing other ROMs, added to that sector with V3 revision of the CL board. *Therefore revisions V3/V4 of the CL board are required.*
- Then there's the question about having to run the programs from RAM for the flash-write/read to work. One could assume that YFNX is already there but it's much better to make sure that's the case by making a copy on the fly and plugging it to the MMU under program control. **Such copy goes to RAM block 0x805**, the Y-Buffer area – **overwriting anything you may have in there previously**. This is especially relevant when using the RAM copy of the IMDB, so make sure that's not the case.
- Note that POWERCL module may reside in Flash during the process, even if the FOCAL program calls upon **YFWR** – as the "from-RAM-only" restriction only applies to YFNS.
- Finally, after the backup copy is made **YWALL** resets the MMU settings plugging the flash copy of YFNX back in. Note that these functions make no assumption on which page YFNX is initially plugged in, and will restore it to the same page after completion. This means it's compatible with any MMU port setting for YFNS – at least one thing not to worry about.

The FOCAL code used by the function is shown in next page – There is also a check done in MCODE looking for the strings "OK" or "OKALL" to be present in Alpha. If neither is there the execution will end with **"NOT OK"** – as a protection against accidental usage. **"OK"** will get the Calculator content backed up, whilst **"OKALL"** will also include the MMU entries into Flash. Note that only the corresponding 4k blocks will be erased, but not the whole 32k sector.

Should any of those default settings clash with your system setup I'd suggest you change it to match them as the easiest way to go around the incompatibilities. Even if it's possible, *re-writing the program in 41-RAM is strongly not recommended.*

1	LBL "YWALL"	
2	IMDBF?	<i>is the IMDB in flash?</i>
3	<i>GTO 01</i>	<i>yes, go on</i>
4	<i>GTO 02</i>	<i>no, skip to end</i>
5	LBL 01	
6	"00F>805"	<i>prepare copy</i>
7	AVIEW	<i>show settings</i>
8	YMCPY	<i>copy YFNS to RAM</i>
9	"805-RAM"	<i>prepare mapping</i>
10	YFNS?	<i>get current YFNS location</i>
11	SF 25	<i>protection override</i>
12	PLUGG	<i>plug RAM there</i>
13	"1FE000"	<i>prepare erasing</i>
14	AVIEW	<i>show settings</i>
15	FC? 49	<i>battery ok?</i>
16	YFERASE	<i>erase BLOCK (!!!!)</i>
17	"800>1FE"	<i>prepare copy</i>
18	AVIEW	<i>show settings</i>
19	YFWR	<i>write calculator to flash</i>
20	FC? 01	<i>was MMU also selected?</i>
21	<i>GTO 00</i>	<i>no, skip to end</i>
22	"1FF00"	<i>prepare address</i>
23	AVIEW	<i>show settings</i>
24	FC? 49	<i>battery ok?</i>
25	YFERASE	<i>erase MMU block</i>
26	"804>1FF"	<i>prepare copy</i>
27	AVIEW	<i>show settings</i>
28	YFWR	<i>write MMU to flash</i>
29	LBL 00	
30	"YFNP"	<i>prepare restore</i>
31	YFNS?	<i>get YFNS location</i>
32	SF 25	<i>protection override</i>
33	PLUGG	<i>plug it there</i>
34	<i>GTO 02</i>	<i>go to end section</i>
35	LBL "YRALL"	
36	"1FE>800"	<i>prepare copy</i>
37	AVIEW	<i>show settings</i>
38	YMCPY	<i>restore backup to RAM</i>
39	"1FF>804"	<i>prepare copy</i>
40	FS? 01	<i>was MMU also selected?</i>
41	YMCPY	<i>restore MMU to RAM</i>
42	LBL 02	
43	"DONE"	<i>show final message</i>
44	AVIEW	<i>and end.</i>
45	END	

Backing up MMU entries may be seen as superfluous, yet think about the issues arising from restoring MMU configurations that don't include the POWERCL module – which is where the program is being run from: welcome to CL-limbo! - Surely something to be avoided.

Similar issues will occur if in the restored configuration the YFNS and POWERCL modules are plugged in a different ports than the ones the restore process is being executed from – the program pointer will get confused and the calculator may hang up.

2.5.6. Transferring ROM images from/to PC files. { DLD48, UPL48 }

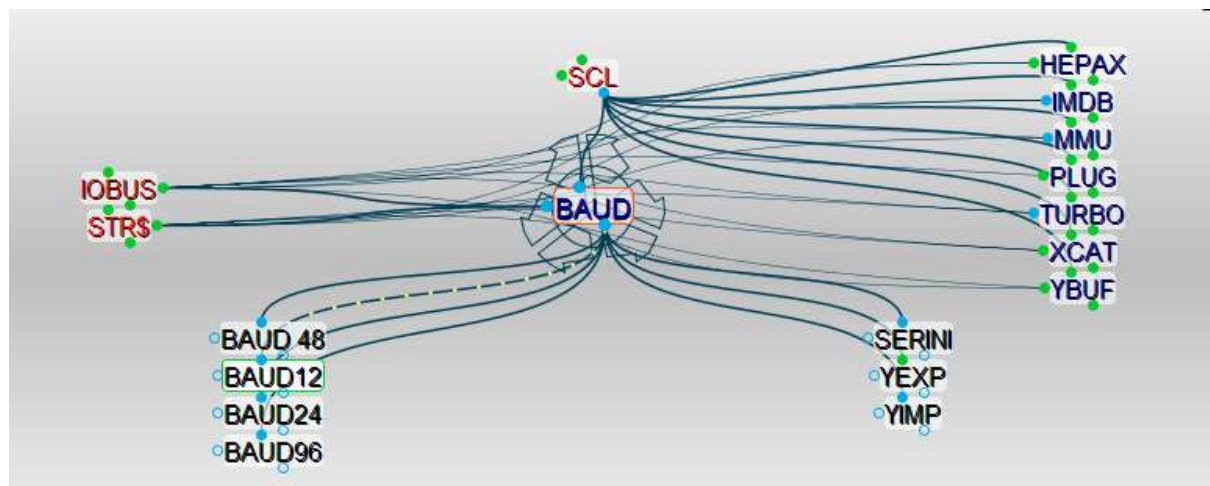
The serial link is a very practical alternative to download and upload ROM images from/to the PC. Definitely worth the set-up and configuration price, even if initially it may look a bit intimidating.

The program below is a nice and short application to automate the downloading and uploading of ROM images from/to the PC, using the serial link. The core of the routine is the function **YIMP**, with the other few lines just setting the stage to make sure nothing is forgotten – typical when done manually. So it's nothing to write home about, but since I use it so frequently I opted for adding them to the module on a permanent basis (got tired of entering the program in RAM every time).

It uses a 4800-baud setting, which happens to be a good compromise between reliability and performance – at least for my own systems. It's very minimalist due to the tightness of space left in the module, but it does its job quite nicely.

1	LBL "DLD48"		11	"ADR: _"	prompt text
2	SF 01	flag case	12	YINPT	input adr
3	GTO 01		13	" -000-0FFF"	control string
4	LBL "UPL48"		14	AVIEW	show feedback
5	CF 01	flag case	15	FS? 01	downloading?
6	LBL 01		16	YIMP	yes
7	SERINI	initialize serial	17	FC? 01	uploading?
8	48		18	YEXP	yes
9	PBAUD	set speed	19	GTO 05	loop back for next
10	LBL 05		20	END	done.

Refer to the YFNS manual for details on the serial link function. Also to appendix 6 for a listing of the CLWRITER and CLREADER, the PC programs running on the other end of the serial link.



CL-to-CL Connectivity.-

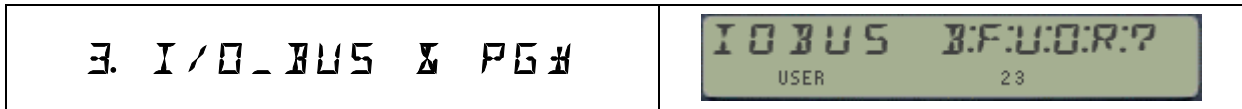
Version 4 of the CL board has the capability to use the serial link without the assistance of a PC connected to it to power the serial bus. This opens the possibility to use a modified cable to link two CL machines together, with the V4 board installed at least in one of them - for the purposes of data and program transferring from one another.

While transferring complete blocks of ram is little more than a trivial exercise (using **YEXP** and **YIMP** on each machine), dedicated software needs to be written to deal with 41 programs or other choices from the actual calculator memory areas. To be continued...



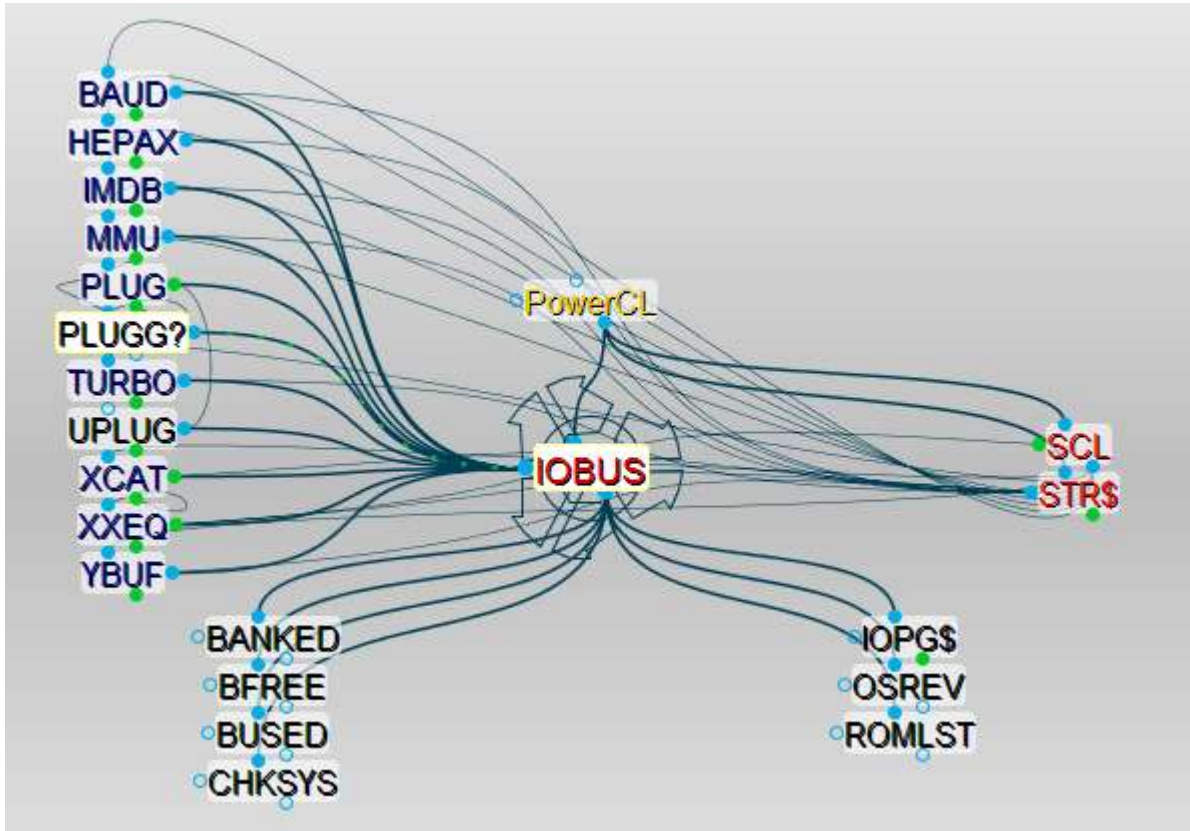
The photo below shows initial experimenting done on this very new capability.





3.1. The system as a whole.

This set of functions provides useful information on the contents of the I/O bus of the calculator – used by the system extensions like plug-in modules, HP-IL, etc. Typically the complete bus is scanned, showing those hits that meet definition asked, such as free or used pages, banked ports, or system configuration checks.



Even if there isn't a dedicated letter for it, the **IOBUS** launcher can be accessed by pressing **[SHIFT]** at any of the launchers prompts (including the main **SCL**). **IOBUS** it's a two-page launcher, with the second part dedicated to Page functions. Use the **[SHIFT]** key to toggle between both:



As always, press **SCL** to return to the main launcher – the real anchor in the UI. The table below lists the functions available at the first page – three of them brand new in revision "J" of the PowerCL:

BANKED	Lists Bank-switched pages	New	
BFREE	Lists Unused pages	New	
BUSED	List pages used	New	
OSREV	Shows OS ROM revisions		Nelson F. Crowle
ROMLST	Lists all ROM id# plugged		
CHKSYS	Checks for ROM id# conflicts		

- **BANKED** presents a colon-separated string of numbers (in hex) corresponding to those pages with a bank-switched configuration, as defined in the ROM signature characters. The official convention is not strictly followed by the (very few) authors of the few bank-switched ROMs, but the number of banks should be marked in characters 2/3/4 of the ROM signature.

An example with both the PowerCL and the SandMath_2x2 plugged returns the following:-
Can you explain the presence of the "5"? Hurry, time's ticking out!



- **BFREE** and **BUSED** will present colon-separated strings of hex numbers corresponding to those free or used pages in the calculator. Obviously the OS will always be listed by **BUSED**, which is a nice clue to quickly tell which particular string you're looking at. See for instance the examples bellow showing a pretty decent configuration:



for the free pages, and



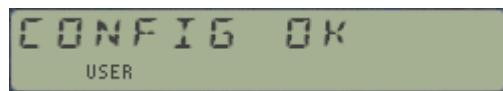
for the used pages.

The strings are compiled using the display, and transferred to ALPHA upon completion. For full-house configurations the list of used pages will take up more characters than those allowed in the display – and the string will be scrolled to the left, dropping the first three pages in the worst case. Since those hold the OS (always there) there's no real information loss.

The strings can have "holes", as this is totally dependent on the modules plugged. Some of them use the upper part of the port (like the Zenrom), or just simply due to the physical locations used.

- **CHKSYS** is a very useful routine to check for incompatibilities in the system configuration, as may occur when two ROMs with the same XROM id# are plugged. The function will scan all the ROM blocks looking for repeat values, showing a confirmation or a warning message depending on the case.

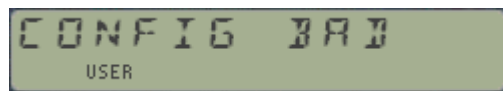
It will also report all and every offending id# in case of conflicts, as many as there may exist. Use it as frequently as you need, it's the best way to ensure that things are fine after plugging any of the many modules available on the CL library – a match made in heaven.



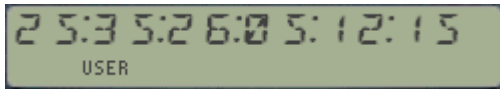
or



plus:



- **ROMLST** has somewhat of a similar purpose: it will produce a list in Alpha with the XROM id#'s of the plugged modules on the system, so you can check for dups. Because of the 24-char limit in the Alpha string, only the last 8 modules will be shown – sufficient in the majority of cases, specially considering that pages 3, 4, and 5 are most likely unique because of being dedicated to the X-Functions, the Library#4, and the Time Module.

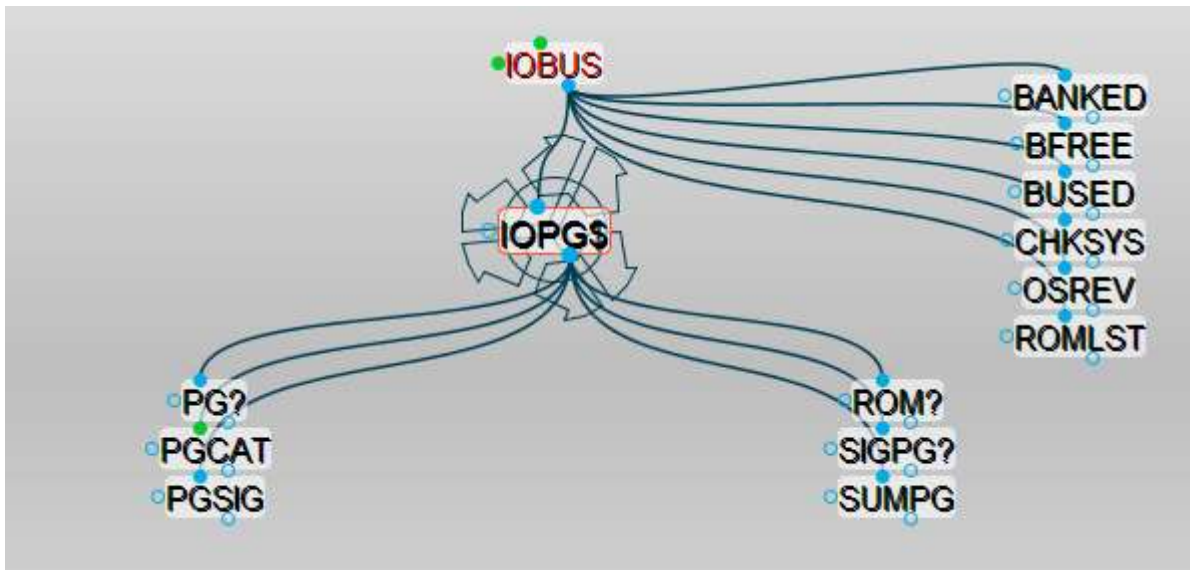


Example: winning Lotto combination or ROM list?

- Lastly, **OSREV** shows the revisions for the three first pages, containing the core Operating System code (in ROMS 1/2/3) / which for an unmodified CL are as follows:



3.2 The Pages within.



Even if these functions aren't strictly new, they have been improved to make them more usable and work in combination with one another within this launcher.

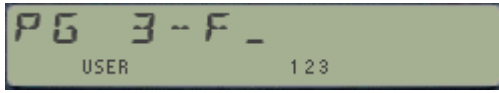
PGCAT	Page Catalog	No Input	VM Electronics
PGSIG	Retrieves ROM signature	Page# in prompt/X	Ángel Martin
SIGPG?	Finds page# if plugged	Signature in ALPHA	Ángel Martin
SUMP	Sums Page checksum	Page# in prompt/X	George Ioannou
PG?	Page vital constants	Page# in prompt/X	W&W GmbH
ROM? _ _	Rom vital constants	XROM id# in X	W&W GmbH

- **PG?** returns miscellaneous information corresponding to the page number input in the prompt in RUN mode, or in X as decimal value if run in a program. The information is as follows:
 - Header function name in ALPHA, and:
 - [XROM id#] ; [# of functions] in X. (in integer and fractional parts)

Note that when used on the POWERCL page it'll return the vital constants for bank-3 (where its code resides), which strangely enough are 56,86 in X (as it was explained at the beginning of the manual)

Considerable trickery has been used modifying this function in revision "J" to be prompting – despite being located in a secondary bank. This makes for a more consistent and usable user

interface, common with other page functions. If there's nothing plugged in the page the message "NO ROM" will be shown.



Input prompt

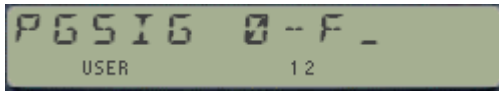


Page is not used (Free).

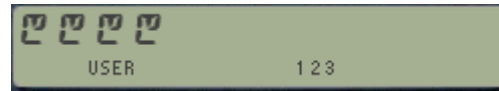
- **ROM?** is also a prompting function. It returns the ROM vital constants for the XROM id# value input in the prompt, as follows:
 - Page# where is plugged in X, and
 - number of functions in Y.

The ROM header (first function name) is also displayed (but not saved in Alpha). Note that this is very similar to **PG?**, only that the input is not the page number but the XROM id# instead. If the ROM is not found the display will simply show "NO" – indicating that this functions doubles as a test function as well, and therefore it'll skip one line in a program in this case.

- **PGSIG** will retrieve the signature string of the ROM plugged in the page entered at the prompt (in Hex format) – or in the X register (in Decimal) if used in a program. If no ROM is plugged it'll return four "@" characters.

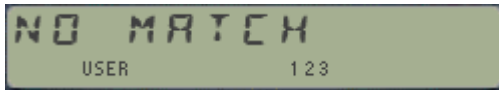


input prompt

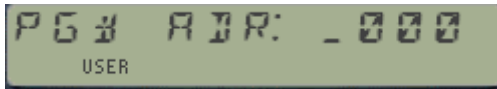
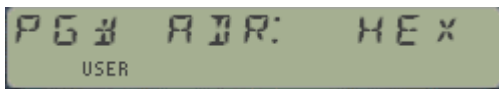


represents a "blank" signature value.

- **SIGPG?** Is the inverse function: it'll return the page# where the ROM with signature input in ALPHA is plugged – returning "NO MATCH" if no matching string is found.

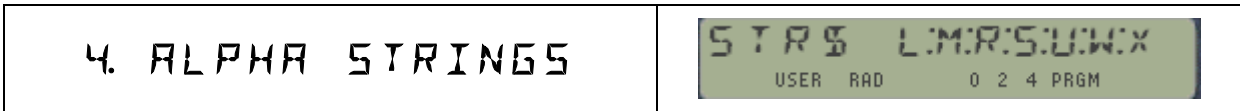


- **SUMPG** prompts for the page number in Hex in a fancy manner, with alternating texts as shown below (that alone covered its admission price). Its mission is to calculate the Checksum byte and to write it in the last word of the page – and that it'll do very nicely.



- **PGCAT** was described before in the CATALOGS section. See below the printout output using JF Garnier's PIL-Box and the ILPER PC program, showing a nice traceability of the pressed keys:

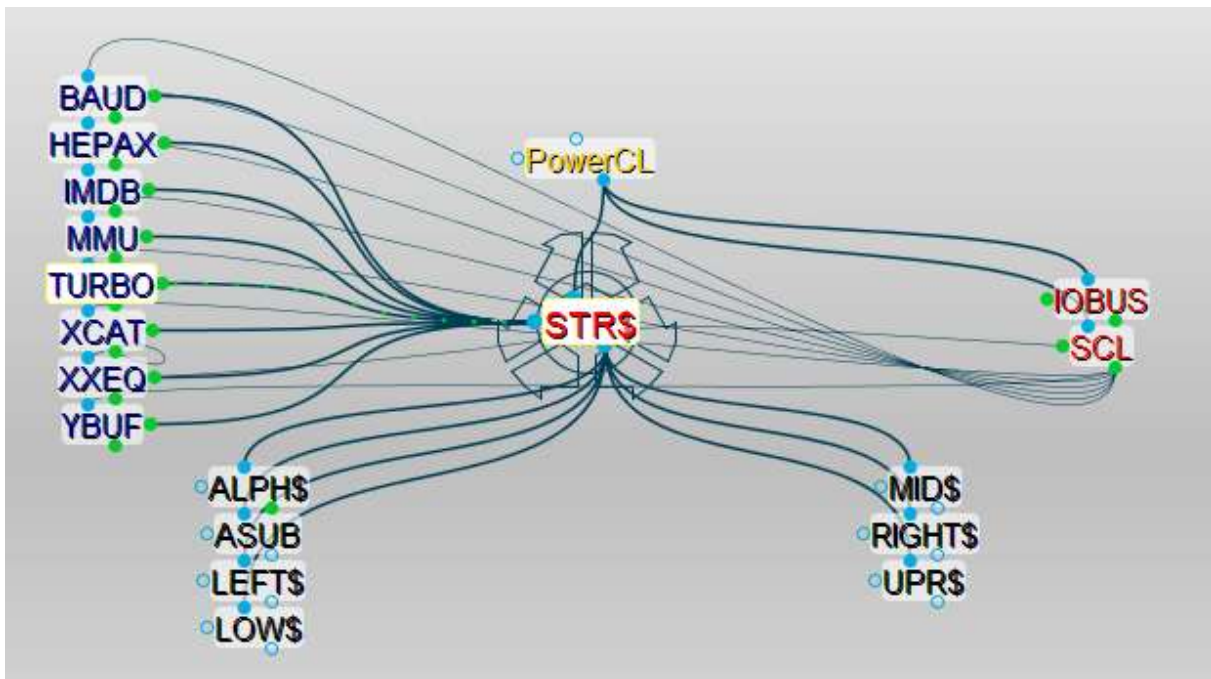
ΣCL / XCAT	9:-TOOLBOX 4
PGCAT	A:-SNDMTH 2X2
3:-EXT FCN 2D	B:-HL MATH+
4:-LIBRARY#4	C: NO ROM
5:-TIME 2C	D:-AMC"OS/X
6:-PRINTER 2E	E:-POWERCL 3X
8:-YFNP'1C	F: NO ROM



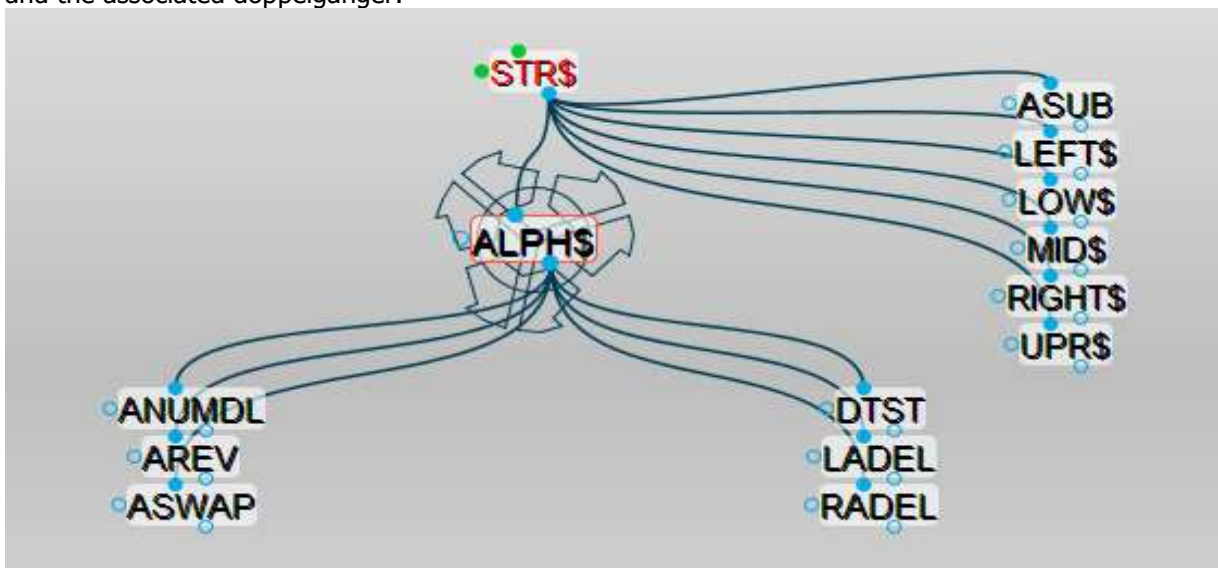
String Manipulation and Alpha/Display functions.

The sub-function groups implemented in auxiliary FATs is a great concept, short from being perfect in all but one component: they cannot be assigned, and thus a full function spelling must be done to invoke them. This can become a nuisance for those sub-functions used frequently, and having them available in a dedicated launcher can be a blessing.

This is exactly the situation with the **STR\$** / **ALP\$** launchers, conveniently accessible simply pressing the **[A]** key at the main launcher prompt. This can be seen in the diagram below, note the related items on the left side:



and the associated doppelganger:



Think of them as ALPHA-Launchers, governed with the [A] key. Pressing [A] again will toggle to the execution of the main Σ CL prompt

4.1. The ALPHA Functions section.

This is now a much-extended set of functions compared to previous revisions. Some are clearly related with one another, and a few of them are symmetrical complements to existing functions in the native set. Let's see them individually with the detail they deserve:-

A<>RG ___	Swap Alpha and Regs.	1 st . RG# in prompt	Ken Emery (original)
AINT	Append Integer part	Takes absolute value	Frits Ferwerda
ALEFTJ	Alpha Left Justify	Moves test to end	Doug Wilder
ANUMDL	ANUM with Deletion	Gets number to X	HP Co.
APPEND	Append Text	Turns ALPHA input on	Doug Wilder
AREV	Alpha Reverse	Mirror image of text	Frans de Vries
ASWAP	Alpha Swap	Around the "," char	Ángel Martin
CHRSET	Character Set	Shows the complete set	Chris L. Dennis
CLA?	Is Alpha Empty?	YES/NO test function	Doug Wilder
DTST	Display Test	Source: PPCJ V18 N8 p14	Chris L. Dennis
DTOA	Display to Alpha	Opposite to AVIEW	Ángel Martin

- **AINT** appends the integer part of the number in X to ALPHA. Perfect to append indexes and counter values without having to change the display settings (FIX 0, CF 29).
- **ALEFTJ** left-justifies ALPHA – by writing as many space characters as required until the total length is 24.
- **ANUMDEL** works like ANUM, but also deletes the number found and placed in X.
- **APPEND** is equivalent to using the "append" functionality manually, i.e. pressing [ALPHA], and "|-" keys. More interestingly, in a program it will prompt the existing ALPHA contents for the user to continue entering more. Similar to **PMTA**, but the initial string will remain.
- **AREV** will reverse the string in ALPHA, turning it into its mirror-image. For instance:



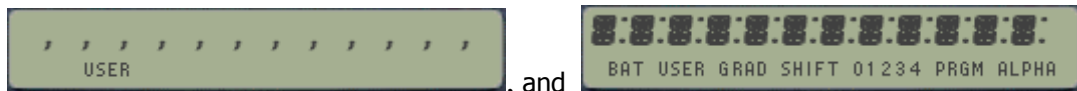
- **ASWAP** swaps the strings at the left and right of the comma character. Very handy for X-Functions data input. Does nothing if comma is not there. For example:



- **CHRSET** will show all characters included in the character set of the machine. There are 128 characters in total, scrolled from right to left in groups of 12. Pressing any key will speed-up the process but the execution won't stop until they're all shown. The screen below shows the enumeration mid-stream:



- **CLA?** Is a test function that returns YES/NO depending on whether ALPHA is empty or not – or in other words, the test is true if **ALENG** is not zero. As usual, in program mode a line is skipped if the test is FALSE.
- **DTOA** captures the display content and writes it into ALPHA. This is an elusive concept, as there are no standard ways to just write text in the display not using ALPHA or other RAM registers – but it's used frequently in MCODE to transfer the display contents to ALPHA.
- **DTST** Simultaneously lights up all LCD segments and indicators of the calculator display, preceded by all the comma characters (which BTW will be totally unnoticed if your CL is running at 50x Turbo!). Use it to check and diagnose whether your display is fully functional. No input parameters are required.



aVIEW	Lower Case AVIEW	No conversion occurs	Ángel Martin
LOW\$	Lower Case convert	Conversion of chars.	Ángel Martin
UPR\$	Upper case convert	Conversion of chars.	Mark Power

- **aVIEW** shows the ALPHA contents displayed all with lower-case characters. Note that there's no conversion to the text in ALPHA, only the displayed characters. Note also that full-nut machines (like those CL donors all are) cannot display most of them, so it's a little academic at this point. On a half-nut machine (or V41) it makes more sense, for instance:



- **LOW\$** does the actual character conversion to lower-case type, thus changing the contents of ALPHA. For the example below, the string is as shown for both the full-nut and half-nut models. This functionality is useful for the printer (which has a complete lower case character set) and when typing ASCII files using lower case characters: just type it "normal", then convert them down in block.



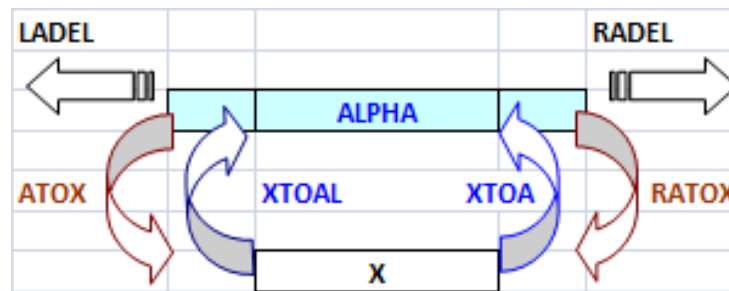
- **UPR\$** does the reverse conversion, getting the text back to upper case characters – perfectly compatible with the machine character set, and thus legible in ALPHA.

ASCII	ASCII value	Does not remove char	Melbourne ROM
LADEL	Left Alpha Delete	Removes leftmost char	Ross Cooling
RADEL	Right Alpha Delete	Removes rightmost char	Ross Cooling
RATOX	Right Alpha to X	Symmetric to ATOX	Ross Cooling
XTOAL	X to Alpha Left	Symmetric to XTOA	Håkan Thörgren

- **ASCII** is similar to ATOX but it does NOT remove the (leftmost) character from Alpha. Use it to read its value into X, but leaving it alone (preserved in ALPHA). The following relationship shall help understand the similarities: **ATOX = ASCII + LADEL**

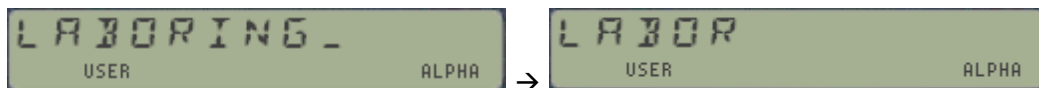
- **XTOAL** appends to the Left of the Alpha text the character which ASCII value is in the X register. This is the symmetrical counterpart to **XTOA**, which appends the character to the right of Alpha. Strictly speaking **XTOA** should be called "XTOAR", and in fact that's how the functions were relabeled in the Extended-IO module.
- **RATOX** removes the rightmost character from Alpha and places its ASCII value in X, also lifting the stack. It is therefore the symmetrical operation performed by **ATOX**, which strictly speaking could be called "LATOX"
- **RADEL** deletes the rightmost character from alpha but doesn't place its ASCII code in X, so it's a "diminished **RATOX**" if you will, in that it only does the first half of it.
- **LADEL** is similar to **ATOX** but not quite the same: Like **ATOX** it will delete the leftmost character from ALPHA, but contrary to **ATOX**, its ASCII value will NOT be placed in X. **ATOX** is a combination of both **ASCII** plus **LADEL** used consecutively; if this comparison helps.

Some of these functions are mutually inverses, one undoing what the other does. The figure below shows these associations more clearly:



POSTSP	Post-space string	Deletes all before SPC	Doug Wilder
PRESP	Pre-space string	Deletes all after SPC	Doug Wilder
ASUB	Alpha Substitute	Replaces character	ZENROM Manual
LEFT\$	Left Substring	Gets left substring	Ross Cooling
MID\$	Middle Substring	Gets Middle substring	Ross Cooling
RIGHT\$	Right Substring	Gets Right substring	Ross Cooling

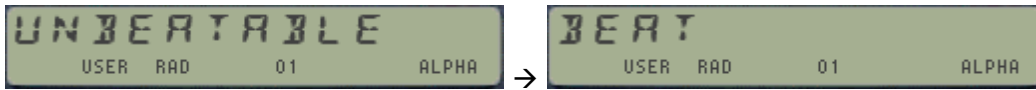
- **LEFT\$** is a sub-string function. They follow the BASIC model, using the number in the X register to define the length of the sub-string. In this case the substring will be taken from the left, truncating it from the rest. For instance with 5 in X:



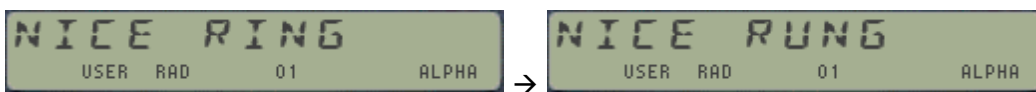
- **RIGHT\$** has the equivalent functionality but taking the substring from the right. The following example should document it, assumes 9 is in X:



- **MID\$** takes the substring from the middle of the text, thus it requires two parameters to define it: the number of characters to remove from the left is expected in Y, and the substring length is expected in X. For example, type: 2, ENTER^, 4, XQ2 "**MID\$**"



- **PRES** and **POST** remove the alpha characters to the right or to the left of the first blank space found, leaving just what was before (PRE) or after it (POST).
- **ASUB** replaces the character located in the *zero-based* position entered in Y with the character which code is in the X register. Only one character at a time will be replaced. For example, with: 6, ENTER^, 85, XQ2 "**ASUB**" will change the text from the left to the right screens:

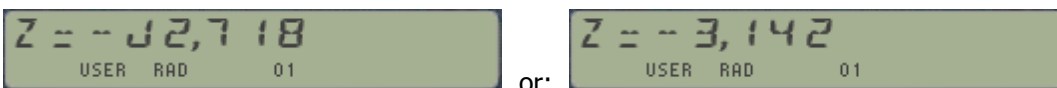


CVIEW	Non-stop VIEW	Avoids the printer halt	Frits Ferwerda
VIEWA	Non-stopping AVIEW	Avoids the printer halt	Ken Emery
ST<>A	Stack exchange w/ Alpha	Register swapping	Ángel Martín
USWAP	Swaps around "<-"	String swapped	Ángel Martín
ASWP>	Swaps around ">"	String in ALPHA	Ángel Martín
YBSP	Alpha back space	Rightmost chr removed	W&W GmbH
YCL>	CLA from ">"	String in ALPHA	W&W GmbH
ZOUT	Complex displaying	Uses X:Y values	Ángel Martín

- **ZOUT** is a complex number display routine. It uses the values in X and Y registers taken as real and imaginary parts of the complex Z, which is presented in the display and ALPHA. The imaginary unit is represented by "J"



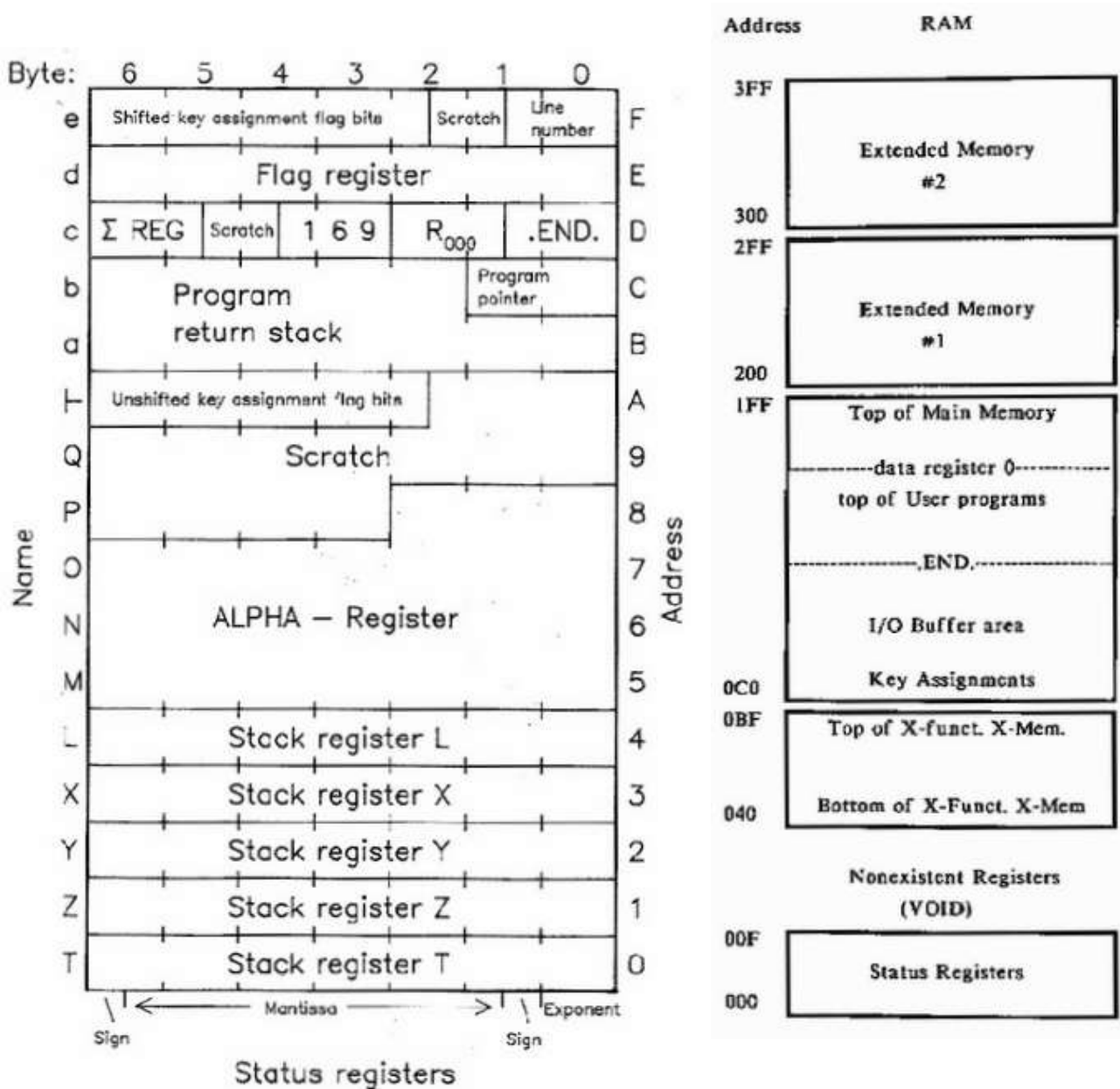
Values are used with the current number of decimal places set (the number returned by **DSP?**), thus the display will scroll if the total length exceeds 12 characters. If any of the parts is zero it'll be omitted from the representation:



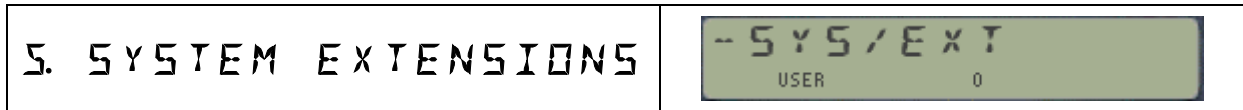
- **ST<>A** and **A<>RG** are simple register exchange routines that swap the contents of the Alpha registers (that is M, N, O, P) with the stack registers X, Y, Z, T or with a register block starting with the RG# input at the prompt respectively. This is handy to temporarily save the stack in alpha for later reuse. Note however that register P is partially used by the OS as scratch, so depending on what you do in between two executions of **ST<>A** the content of the T register may have changed.

- **ASWAP**, **YCL>**, and **YBSP** are pretty much self-explanatory, performing partial Alpha deletions and text swapping around the control characters "<->" and ">". They are used by some of the FOCAL programs included in the module. Because they were used in FOCAL programs, these three functions were in the main FAT in previous revisions of the PowerCL module, but have been moved to an auxiliary FAT after the program was modified.

This is probably a good moment to be reminded about the register map of the calculator, including the buffer area (relatively small in comparison), the Status registers and the complete extended memory – see figures below, taken from the CCD Manual and "M<CODE for Beginners" book: (*)



(*) Correct addresses should be 201-2EF and 301-3EF for the first and second EM modules.



5.1. X-Memory Utilities.

The table below shows the advanced X-Memory utilities, complementing and enhancing the standard capabilities of the native system. Usual suspects, you'll surely recognize the names.

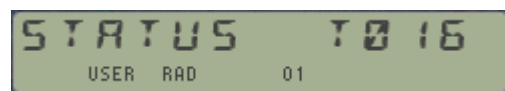
ARCLCHR	ARCL Character from file	FNAME in Alpha	Håkan Thörngren
CLEM	Clear Extended Memory	No inputs	Håkan Thörngren
CLRAM	Clear all RAM	No inputs	R. del Tondo
FHD	File Header	FNAME in Alpha	Ángel Martin
FLTYPE	File Type	FNAME in Alpha	Ángel Martin
PEEKR	Peek Register	RG# address in X	Ken Emery
POKER	Poke Register	RG# address in Y	Ángel Martin
RENMF	Rename File	OLDName , NEWName	Ángel Martin
RETPFL	Retype File	NAME in Alpha, type in X	Ángel Martin
RSTCHK	Reset Checksum	FNAME in Alpha	Håkan Thörngren
XQXM	Execute (Program) File	FNAME in Alpha	Ross Cooling
READXM	Read X-Mem from Disk	FNAME in Alpha	Skwid
WRITXM	Write X-Mem to Disk	Disk FNAME in Alpha	Skwid
GETST	Get Status Registers	FNAME in Alpha	Ángel Martin
SAVEST	Save Status Registers	FNAME in Alpha	Ángel Martin
WORKFILE	Appends Workfile name	none	Sebastian Toelg

The following short descriptions summarize the most important points for each function:

- **ARCLCHR** appends the character at the current pointer position of the ASCII file which name is in Alpha (or the current file if no name is provided). The Pointer is advanced one position, ready to retrieve the next character if needed.
- **CLEM** is an expeditious Extended Memory eraser – all files will be gone, just by deleting the content of the X-Mem main status register, at address 0x040. In RUN mode the function will show the message "DIR EMPTY" for confirmation.
- **CLRAM** will go beyond the previous scope, also clearing the contents of Main memory: Buffers, Key assignments and data registers – in sum, ALL RAM will be gone. Similar to MEMORY LOST for all that counts, so use it at your own risk.
- **FHD** will return the absolute address (in decimal) for the Header register of the file named in Alpha. This is useful as input for **PEEKR** and **POKER**, **RAMED** and other memory editing functions.
- **FLTYPE** returns the type of the file which name is given in Alpha. Valid file types are shown in the table below, note the five custom extensions supported by the AMC_OS/X module:

File	PRGM	DATA	ASCII	Matrix	Buffer	Keys	"T"	"Z"	"Y"	"X"	"H"
Type	1	2	3	4	5	6	7	8	9	10	11

- **RENMF** is a handy utility that renames an X-Mem file. The syntax is the same used by **RENAME** for the HPIL Disks, that is the string "**OLDNAME,NEWNAME**" must be in alpha. The function will check that the OLDNAME file exists ("**FL NOT FOUND**" condition otherwise), and that there isn't any other file named NEWNAME already ("**DUP FL**" error message).
- **RETF** is a bit of a hacker trick: it modifies the file type information for the file named in Alpha, changing it to the value in X. This is actually useful in a number of circumstances, like sorting a Matrix file using **SORTFL** (which only works for DATA files): just change the type to "**2**", sort its contents with **SORTFL**, and change it back to "**4**". You can use any value from 1 to 14 in X, other values will cause "**FL TYPE ERR**" conditions
- **RSTCHK** is a rescue function that restores the checksum value for a PROGRAM file. Use it if this byte gets corrupt or when you alter the program file manually (hacker beware!), so the file will recover its "valid" status.
- **XQXM** is a PROGRAM File Execute - direct execution of the program. Note that all GOTO's must be pre-compiled, and no calls to other programs may exist within the file.
- **WRTXM** and **READXM** are used to write/read the complete contents of the X-Memory to/from a disk drive over HPIL. These functions exercise the full capability of the system, and provide a nice permanent back-up for your XMEM files. Note that only the non-zero content will be copied, thus the resulting disk file size will not be larger than required - in other words, it won't always copy all XMEM even if zero, like other FOCAL implementations of the same functionality can only do. These functions are taken from the Extended-IL ROM, written by Ken Emery's alter ego Skwid.
- **WORKFL** will append the name of the current workfile to ALPHA. Easy does it!
- **SAVEST** and **GETST** are special in a couple of ways. For starters because their subject is the complete Status Registers, i.e. the "Chip0" of the system RAM. Use **SAVEST** to make backups of the entire status registers area to XMEM, including the stack, flags, Alpha, and the other control registers. Use **GETST** to restore the status registers back to the same state. For obvious reasons the file size will always be 16. They're also special because they use a file type Z, which is properly recognized as type "**T**" by the CAT'4 implementation in the AMCOS/X module:

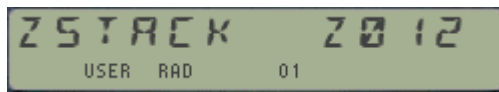


A couple of observations are in order:

- The XMEM file name is expected in ALPHA, thus this imposes a little limitation on things. You can however add a comma to the FileName and write additional text after it - which will be ignored by the functions.
- Register 12(b) stores the program counter (PC). Executing **GETST** in a program will overwrite the current PC, and the program execution will be "lost" - going to the same place it was at when the status registers were saved. There are more tricky issues using these in PRGM mode, like the question of the subroutine stack and the program line. Suffice it to say it's not really advisable - yet I resisted the idea to make it non-programmable, but users beware!

- Saving and restoring the Key Assignments involves two separate actions. **GETST** only restores the key mappings in registers 10(|-) and 15(e), but it doesn't have anything to do with the actual KA registers in the I/O area. Make sure you use **SAVEKA** and **GETKA** instead for this need, or the key assignments will be scrambled. See the KA utilities section.
- **SAVEZS** and **GETZS** are really buffer functions in disguise – thus a good segue way to the next section. Use them to store and recall the Complex stack to/from an Extended Memory file. This includes the five complex registers Z, W, U, V, and LastZ; plus the buffer header and control registers – thus the file size is always 12. (Refer to the 41Z manual for details if interested).

The file type used is 8, which is recognized by the CAT'4 implementation in the AMCOS/X as type "Z" (how could it be otherwise?):



Note: For the Z-Stack you can also use **SAVEBF** and **GETBF** with a buffer id#=8 input in X (see next section), but the resulting XM-file will have a generic type "B", as opposed to the specific for the Complex Stack.

- Last in this section are well known amongst all HP41 users, PEEKR and POKER. **PEEKR** can be compared to the RCL function, however it is now possible to read the contents of any register, and without normalization, into the X register. This removed one of the main problems of synthetic programming. The address of the register to be read is entered as absolute address in to X. As when using RCL IND X, the stack registers are lifted. **PEEKR** works for every existing register address from zero to 1,023. If we want to use relative data register numbers with **PEEKR**, the absolute address of the data registers must be first obtained.
- **POKER** writes over the absolute register, which address is specified in the Y register, with the contents of the X register. **POKER** works for the entire existing register range of the calculator. The stack registers remain unchanged, as long as they are not specified by the absolute address in Y. Since **POKER** can change any register, this function should only be employed if the calculator structure is well understood. Otherwise it may result in unwanted changes in programs, data registers, status registers, etc. or even a MEMORY LOST condition.

Example: Creating second sets of Main and Extended Memory.

A nice utilization of these functions on the 41CL are the examples shown below to create backups of your complete Main memory and Extended memory sets – located in RAM block 0x801 (that is, above the standard calculator RAM space located at 0x800).

Because **PEEKR** and **POKER** accept input addresses higher than the standard calculator range, they're well suited for the task. Basically all we need to do is copy the contents of the Main/Extended memory from its current addresses (refer to figure in page 66) to addresses located 1k above them. In fact, one can have an alternate set of memory and "swap" between both as needed, duplicating so the calculator's on-line capacity. An additional benefit is that *the secondary set will not be affected in case of a MEMORY LOST*, thus you can use it as a safety backup as well.

Main memory is trickier than extended in that the status registers should also be included in the backup to ensure a properly configured FOCAL chain and memory configuration. These **must** include register 13(c), and ideally also 10(+), 14(d) & 15(e) for compatible flags and key assignments definition (together with the KA registers in the I/O area).

Below are the programs to swap the sets of Main and Extended memory at your convenience, **MMSWAP** and **XMSWAP** respectively:

1	LBL "MMSWAP"		36	239	extended size
2	CF 00		37	X<> M	store in M
3	E	register 10(+)	38	LBL 03	
4	STO M		39	RCL X	origin
5	10	reg(+) address	40	1024	offset address
6	XEQ 03		41	+	destination
7	3	registers b, c, d, and e	42	LBL 01	
8	STO M		43	PEEK R	read destination value
9	13	reg(b) address	44	X<> Z	
10	XEQ 03		45	PEEK R	read source value
11	"MAIN"		46	R^	lift for compare
12	AVIEW		47	X=Y?	are they equal?
13	320	all memory	48	SF 00	flag it so
14	STO M		49	RDN	undo lift
15	192	origin	50	FS?C 00	equal values?
16	GTO 03		51	GTO 02	yes, skip writing
17	LBL "XMSWAP"		52	X<> T	position them crossed
18	"XF/M"	base block	53	POKER	write them back
19	AVIEW	provide feedback	54	RDN	
20	CF 00	base block	55	X<> Z	position them crossed
21	64	origin	56	POKER	write them back
22	XEQ 00	swap block	57	LBL 02	merge code
23	"EM-1"	first EM module	58	RDN	locate addresses in X, Y
24	AVIEW	provide feedback	59	E	
25	SF 00	larger block	60	ST+ Z	increase adr-1
26	513	origin	61	+	increase adr-2
27	XEQ 00	swap block	62	DSE M	decrease counter
28	"EM-2"	second EM module	63	GTO 01	loop back if not done
29	AVIEW	provide feedback	64	"DONE"	block is done
30	SF 00	larger block	65	AVIEW	
31	769	origin	66	END	
32	LBL 00	swap block			
33	X<> M	address to M			<i>*always* use MMCOPY before the first time</i>
34	128	base size			<i>using MMSWAP to ensure proper status regs values</i>
35	FS?C 00	larger block?			

The program below copies the main memory to the higher location for a backup, or to prepare the destination for successive main memory swapping (needs to "prime the pump" to make sure the second set is compatible with the OS).

Note: you could also do the initial step copying the complete 4k block using YMCPY, with the following string in ALPHA: "800>801". This would be faster than MMCOPY but will not discriminate Main Memory from Extended one, so both will be backed up.

*Note: this section is clearly superseded by the function set described in the **YMEM** launcher, see section 4.1, pages 43 to 45. It has however some usefulness as it only requires OS/X functions and thus can be used independently from the PowerCL itself.*

1	LBL "MMCOPY"		19	1024	
2	10	reg(+) address	20	X<>Y	
3	PEEKR	read current	21	+	adress-2
4	1034	destination adr	22	LASTX	
5	X<>Y		23	LBL 01	
6	POKER	copy value	24	PEEKR	read value in adr-1
7	13	reg(c) address	25	X<>Y	
8	ENTER^		26	RDN	
9	3	registers c, d, and e	27	POKER	write in in adr-2
10	XEQ 03	copy block	28	X<> T	
11	"MAIN"	main memory	29	E	
12	AVIEW	provide feedback	30	ST+ Z	increase adr-1
13	192	origin address	31	+	increase adr-2
14	ENTER^		32	DSE M	decrease counter
15	320	number of registers	33	GTO 01	loop back if not done
16	LBL 03		34	"DONE"	block is done
17	STO M	save counter in M	35	AVIEW	
18	X<>Y	address-1	36	END	

At the risk of stating the obvious, **MMSWAP** cannot be run from main memory! – or you’ll get nice pyrotechnics and a guaranteed ML event. Make sure you run it from ROM (HEPAX or similar), or even from X-Memory if you are up to those tricks.

As interesting as these examples are, don’t forget that being in sRAM the alternate sets of Main and Extended memory will be lost when the calculator battery goes flat. For a permanent backup refer to section 2.5.5 on page 53, describing the flash backup and restore programs **YWALL** and **YRALL**.



CL board: The soul in the machine.

5.2. Buffer Utilities

Fascinating things these Buffers, so challenging and elusive they are that prompted the development of the **BUFFERLAND** Module. Many of its functions are incorporated in here as well, as follows:

BFLNG	Buffer Length	Buffer id# in X	Ángel Martin
BFLST	Lists Buffers	Shows list in Alpha	David Yerka
BFVIEW	Views Buffer contents	Displays all registers	Ángel Martin
BUFHD	Buffer Header	Address to X	Ángel Martin
CRBUF	Creates Buffer	ID#,SIZE in X	Ángel Martin
DELBUF	Deletes Buffer	Buffer id# in X	Ángel Martin
GETBF	Gets Buffer from file	FileName in Alpha, id in X	Håkan Thörngren
GETZS	Gets Z-Stack from file	FileName in Alpha	Ángel Martin
REIDBF	Re-issue Buffer id#	OLDID,NEWID in X	Ángel Martin
RESZBF	Resize Buffer	ID#,SIZE in X	Ángel Martin
SAVEBF	Save Buffer to File	FileName in Alpha, id in X	Håkan Thörngren
SAVEZS	Saves Z-Stack to File	FileName in Alpha	Ángel Martin

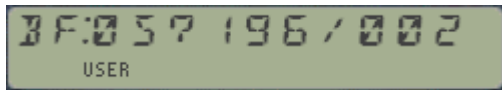
Not listed here is the "queen" buffer function, **BFCAT** – which was included in the "[CATALOGS](#)" section covered before. A quick summary recap on "buffer theory" will help understand this section better:-

1. Buffers reside in the I/O area of RAM, which starts at address 0x0C0 and extends up until the .END. register is found. Typically they are located right above the Key Assignments registers, the only exception being buffer-14, used by the Advantage Pac to hold the **SOLVE** and **INTEG** data (expected to be in fixed addresses by the code). Note that this implies that *the actual location of a buffer will be dynamically changed* when Key assignments are made or removed; when timer alarms are set or run, and possibly also when other buffers are removed - either by the OS housekeeping tasks or using the buffer functions.
2. Each buffer has a header register (at the bottom) that holds its control data. The structure of the header varies slightly for each buffer, but all must have the buffer type (a digit between 1 and E) repeated twice in nybbles 13 and 12, as well as the buffer size in nybbles 11-10 (maximum 0xFF = 255 registers). The rest are buffer-dependent; for example the 41Z buffer holds the data format (RECT or POLAR) in nybble 9. The HP-IL Devel buffer uses nybbles 9-7 to store the pointer value, and nybble 3 to hold the pointer increment type (MAN or AUTO).

T	T	S	Z								A	D	R
13	12	11	10	9	8	7	6	5	4	3	2	1	0

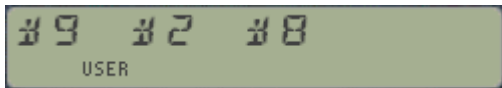
3. Some buffers write the address location in the [S&X] field (nybbles 2-0) but this is of relative use at most, since the buffer can get re-allocated as mentioned above. In fact, **BFCAT** uses that field to record the distance to the previous buffer in the I/O area, necessary to keep tabs with the RAM structure in **SST/BST** operation mode.
4. When the calculator awakens from Deep Sleep the OS erases nybble 13 from all buffer headers found. The execution is transferred to the Polling Points of those modules present, which should re-write the buffer type in that nybble for those buffers directly under their responsibility. At the end of this process (when all Modules have done their stuff) the OS performs a packing of the I/O area, deleting all buffers not preserved" – i.e. with nybble 14 still holding zero.
5. Under some rare circumstances a given buffer can exist in memory as a "left-over" not linked to any module (i.e. nybble 13 in the Header register is cleared). The OS upon the next PACKING operation will reclaim these orphan buffers, so their life span is very short – get what you need from it before it's gone!

Note that to denote this contingency, **BFCAT** will add a question mark to the buffer id# in the display. For example, see the screen below showing an "orphan" buffer id#5 on V41:



With these preambles made let's delve into the description of buffer functions. The following general remarks and individual comments apply:-

BFLST is a poor man's version of the buffer catalog, showing a short list in Alpha of the buffer id#'s currently present in the system; (lean and compact, see example below):



When the function operates on a given buffer its id# is expected to be in the X register. This is the case for **BFLNG**, **BUFHD**, **BFVIEW**. The X-MEM Save/Get functions **SAVEBF** and **GETBF** (seen in next section) also expect the File Name in Alpha.

The input for **CRBUF** and **RESZBUF** is given in X as a combined decimal number: ID#,SIZE The integer part represents the buffer id# (must be between 1 and 14), and the decimal part its size (must have three decimal places). Maximum size is 255 registers, larger values (as well as zero) will trigger "DATA ERROR" messages. This convention also applies to **REIDBF**, so the new id# must be expressed with three significant digits:

9,010 ; XEQ "**REIDBF**" -> changes buffer #9 into buffer #10

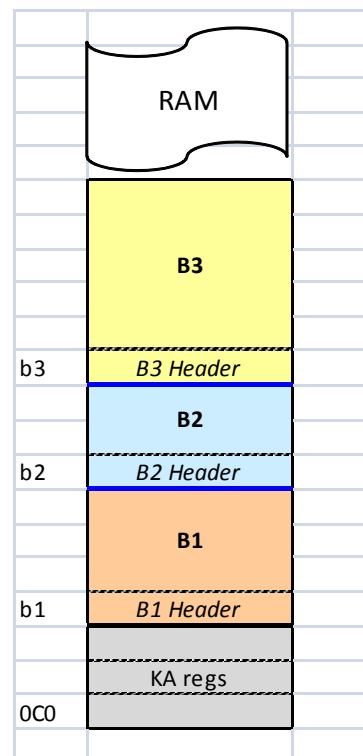
Note that those buffers created with **CRBUF** are somehow "extemporaneous" (i.e. not managed by dedicated modules) - thus they'll be short-lived by nature, because they won't survive a power-off cycle.

The operation of **RESZBF** is compatible with KA and multiple buffers existence. Note however that while upsizing a buffer will be smooth and will keep the previous buffer contents, *downsizing it will cause loss of the data contained in the removed section.*

DELBUF will remove the buffer with id# in X. It works simply by clearing the nybble 14 in the buffer header register, and then calling the OS routine [PKIOAS] to reclaim the registers previously used by it – so no "uncommitted" buffers are left behind. This function is equivalent to **CLB**, available in the CCD Module and its derivatives (like the AMCOS/X). Also available in those ROMs is function **B?**, to test the existence of a buffer.

Saving and Getting buffers in/from Extended memory with **SAVEBF** and **GETBF** follows the same convention used for other file types, with the buffer id# in X and the FileName in Alpha. Error handling includes checking for duplicate buffer ("DUP BF"), buffer existence ("NO BUF"), as well as previous File existence ("DUP FL"). It is possible to have multiple files with the contents of one specific buffer id#, but only one buffer id# can exist in the I/O RAM area at a time.

GETBF, **CRBUF** and **RESZBF** will check for available memory, showing "NO ROOM" when there isn't enough room in main RAM to proceed.

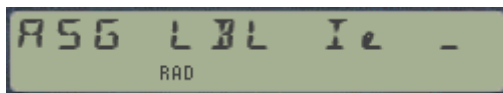


5.3. Key Assignment Utilities

The table below shows the **Key Assignments related functions**. Typically no inputs are required (no need to identify the "buffer type" in this case), with a few of exceptions:

ASG	Multi-byte ASN	Supports synthetics	Frits Ferwerda
GETKA	Gets KA from File	FileName in Alpha	Håkan Thörngren
KACLR	Clears KA / Bufefrs	OK/OKALL in Alpha	Hajo David
KALNG	KA Length	No inputs	Hajo David
KAPCK	Packs KA Area	No inputs	Hajo David
LKAOFF	Deactivates Local KA	No inputs	Gordon Pegue
LKAON	Activates Local KA	No inputs	HP Co.
MRGKA	Merge KA to File	FileName in Alpha	Håkan Thörngren
SAVEKA	Save KA to File	FileNAme in Alpha	Håkan Thörngren

- **ASG** is another example of first-class MCODE programming: imagine being able to directly input multi-byte functions (even with synthetics support) into the ASN prompt, so to assign "LBL IND e", or "RCL M" to a key - not using key codes or byte tables? Well no need to imagine it, just use **ASG** instead. This function is taken from the MLROM, and it resides completely in the Page#4 Library, with only the FAT entry in the POWERCL calling it. You're encouraged to refer to the MLROM documentation for further details.



- **KACLR** expects a confirmation string in Alpha. With "OK" only the KA will be erased, and with "OKALL" the complete IO area will be purged (that is the KA registers and ALL buffers).
- Saving and getting KA in/from Extended Memory with **SAVEKA**, **GETKA** and **MRGKA** also expects the FileName in Alpha. **GETKA** will completely replace the existing key assignments with those contained in the file, whilst **MRGKA** will merge them – respecting the unused keys so only the overlapping ones will be replaced. Same error handling is active to avoid file duplication or overwrites. Like their Buffer counterparts they will check for available memory, showing "NO ROOM" when there isn't enough for the retrieval.
- **LKOFF** and **LKON** are meant to be used together, to temporarily suspend the local key assignments (on keys A-J) so that it doesn't interfere with local program labels. These functions only modify the key mappings in status registers 10("I-") and 15("e"), not altering the actual KA registers.
- **KAPCK** will pack the KA registers, compacting the voids (blanks) left behind when un-assigning individual function keys. The diagram below shows that each KA register can hold up to four key assignments, two nybbles for the key code and two for the function id#. It also shows that they always have 0xFF in nybbles 13 and 12 – which explains why 15 is not available for buffer id#.

F	F	K	Y	C	D	K	Y	C	D	K	Y	C	D
13	12	11	10	9	8	7	6	5	4	3	2	1	0

- **KALNG** returns the number of registers used in the I/O area for key assignments. So you could use it before and after calling **KAPCK** to see the effect of the packing (if anything at all). If no key assignments exist then the result will be zero.

5.4. Page Functions Revisited.

The following group deals with more page-related functions not covered in the **IOPG#** launcher section. These are loosely defined as those functions operating on complete pages, or retrieving page-related information. They are as follows:

BANKS?	Gets # of banks	Page# in X (decimal)	Ángel Martin
BLANK?	Checks page contents	Page# in X, YES/NO	Ángel Martin
CLBG	Clear Block	BBBEEE in X (as NNN)	Frits Ferwerda
PGCPY	Copy Page	FROM: in X, TO: prompt	Ángel Martin
PGROOM	Counts zero words	Page# in X (decimal)	Ángel Martin
PGSUM	Sums Page checksum	Prompts for Page#	Frits Ferwerda
READPG	Reads page from HP-IL	Page# and FileName	R. del Tondo (?)
ROMCHKX	Verifies ROM checksum	XROM id# in X	HP Co.
WRTPG	Writes page to HP-IL	Page# and FileName	R. del Tondo (?)
X=PG?	Compares two Pages	Pg# in X and Prompt	Ángel Martin
DECODE	NNN to HEX string	NNN in X	VM Electronics
CODE	HEX string to NNN	Hex string in Alpha	VM Electronics
XQ>XR	XEQ to XROM	Converts instructions	W&W GmbH

Note the color-code convention for function names: **BLUE** means located in the main FAT (use XEQ to invoke them), **BROWN** means located in a secondary FAT (i.e. you need to use **XQ1** or **XQ2** to invoke them). Of course you can *always* use **XQ2** for any of them, as discussed before.

BLANK?, **BANKS?**, and **PGROOM** were added in revision "J", and (even if they are sub-functions!) are prompting as of revision "N2". The three share the same data input approach: a hex page id# (0-F) in RUN mode, and a decimal number in X in program execution.

- **BLANK?** Is a test function that checks the contents of a full page, looking for non-zero words, displaying YES/NO in RUN mode accordingly. If at least one word is not a zero the result is false and a program line is skipped when used in a program. Note that the word FFF is also considered to be a blank; this is used by the CL and some other MLDL boxes for "empty" Flash blocks.
- **BANKS?** returns the number of banks for a given page, which number in decimal is input in X. The allowed range of results is of course 0-4: non bank-switched ROMS return a "1", and empty pages will return zero. This function reads the third nibble of the last three words in the ROM signature, which is where the bank-switched configuration is supposed to be recorded according to some undocumented criteria. The few authors who released this type of modules followed this rule loosely, thus the result may be a little off. For example, the convention used by Zengrange for the ZEPROMs is not exactly the same.

Not even HP followed this to the letter, or if they did I cannot figure out the Advantage's scheme. Another discrepancy occurs on the CL itself, where the time signature of the TIME module has been altered – misleading **BANKS** to report 4 banks instead of just two.

- **PGROOM** counts the number of words with zero value in the page which number is in X. Interesting to see the density of your favorite MCODE modules (use the OS as a ranking benchmark), and to get an idea on how much room is still available in the page.

Application Example.- How big is your lollypop?

The short program below – **XRAY** – will calculate the complete number of banks configured on-line in the calculator at a given time. I have corrected the TIME Module glitch just by starting to scan the I/O but at page 3, thus the extra banks reported for it would account for the OS pages 0-2.

The program first enumerates the banks found for each page (set it in **TURBOX** if you want to see them), and then shows the total actual size, given in kilobytes - with 4k per page. Note that the listing shows **BANKS?** and **ARCLI** for clarity, but they are in fact **XQ2** and **XQ1** functions with their corresponding indexes in a second program line.

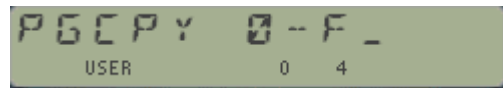
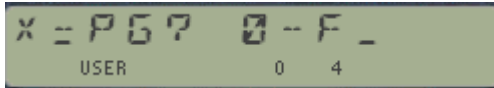
1	LBL "XRAY"			12	RDN
2	0			13	ISG X
3	3,015			14	GTO 00
4	LBL 00			15	X<>Y
5	"PG#"			16	4
6	ARCLI			17	*
7	" :-"			18	CLA
8	BANKS?			19	ARCLI
9	ARCLI			20	" -K"
10	AVIEW			21	AVIEW
11	ST+ Z			22	END

Examples. An MMU-disabled CL would return 24k for the CX OS plus another 4 for the YFNS/YFNP; that is a minimum "baseline" of 28k. Obviously the PowerCL must be plugged in as well to run the program, which adds another 16k to the sum you're likely to see,

A full-house configuration like the one shown in the figure below can have up to **132 kB**, quite an impressive feat considering we're talking about a hand-held calculator design from 1979 – which although extended, expanded, and stretched to the limit really shows the versatility and solid engineering of the design.

Port	Page	Addresses	Primary Bank	Secondary Bank	Bank #3	Bank #4
4	F	FFFF F000	Hepax RAM			
	E	FFFF E000	HEPAX_1D- b1	HEPAX_1D- b2	HEPAX_1D- b3	HEPAX_1D- b4
3	D	DFFF D000	ADV Matrix - B1	ADV Matrix - B2		
	C	CFFF C000	SandMatrix - B1	Vector Calc- B2		
2	B	BFFF B000	HL_Math - B1	HL_Math - B2	Solve & Integ	
	A	AFFF A000	SandMath - B1	SandMath - B2	AEC Solvers	
1	9	9FFF 9000	POWERCL-B1	POWERCL-B2	POWERCL-B3	POWERCL-B4
	8	8FFF 8000	YFNP_1C			
hpil	7	7FFF 7000	AMCOSX-4	AMCOSX4 - B2	AECPROG - B3	
	6	6FFF 6000	PRINTER	IR Printer - b2		
	5	5FFF 5000	TIMER	CX FNS - Bank 2		
	4	4FFF 4000	Library #4	CL Library		
	3	3FFF 3000	CX FNS- Bank 1			
	2	2FFF 2000	OS - ROM 2			
	1	1FFF 1000	OS - ROM 1			
	0	0FFF 0000	OS - ROM 0			

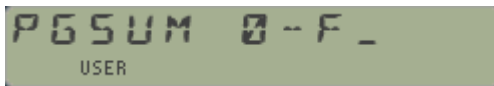
- **CLBG** will clear the block between addresses "bbbb|nnnn" given as a NNN placed in X, which is used as input. If the input is just one digit it'll delete the complete page. Obviously will only work with RAM-mapped pages. Note that **CLRAM** in the "-HEPAXA2" section also clears the complete page, and it takes a decimal input for the PG# in X. Both require the string "OK" in Alpha, as a security measure to avoid accidental usage.
- **X=PG?** compares the complete contents of two pages, returning YES/NO depending on the result and skipping a line in PRGM mode when false. The execution time will depend on the logical condition, whereby a "NO" result is returned as soon as the first non-equal word is found.



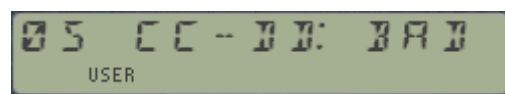
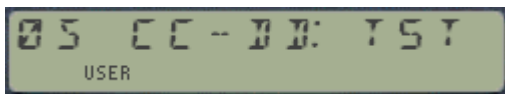
- **PGCPY** copies the contents from page number in X to the page entered in the prompt (in hex format, 0-F). The destination page must be mapped to RAM for a correct operation. No confirmation string is required!

COPYROM in the "-HEPAXA2" section, pretty much does the same as **CPYPG**, but taking the FROM: page input from Y and the TO: page input from the X register instead of the prompts. The function name is somehow misleading, as it's operating on PAGE numbers and not on XROM id#'s – I left them unchanged as in the original HEPAX.

- **PGSUM** does the same as **SUMPG** (seen in the IOPG# launcher section) but using a simpler prompting approach for the input. In program mode **PGSUM** will take the page number from X. Its mission is to calculate the Checksum byte and to write it in the last word of the page – and that it does very nicely. And nothing better than using the next function to verify the result...



- **ROMCHKX** will check the ROM which XROM id# is in X for the correct checksum byte value. The display shows information message while the test takes place, followed by a confirmation or a warning depending on the case.

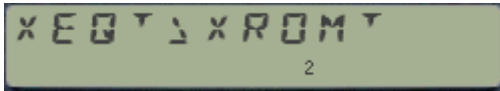


Incidentally it's more than likely that if you run **ROMCHKX** on the POWERCL itself the result is "BAD". This is not because of an error; I just usually don't bother to update the checksum values, as the code is updated very frequently.

- **READPG** and **WRTPG** are the mandatory read/write entire blocks (a.k.a. pages) to the HP-IL disk drive. Very much equivalent to the HEPAX' **READROM** and **WRTROM**, where the destination page is expected to be in X. It works on any page, RAM or ROM, and OS included. Note: for bank-switched modules only the first bank is copied!

Their code is entirely contained in the Library#4, so this is another example of the "free-riders" only needing the FAT entry and the calling stub footprint. They are taken from the CCD OS/X, thus I attributed authorship to R. del Tondo – which to this date is unconfirmed.

- XQ>XR** is without a doubt a powerful function. It converts the **XEQ** instructions included in a FOCAL program (saved in Q-RAM) into the appropriate **XROM** equivalents, assuming that the calls were made other programs residing in a plugged-in module. The need for this arises when programs are loaded on Q-RAM devices, like the HEPAX RAM. The net result is substantial byte savings, because any XROM instruction takes only two bytes, regardless of the label length of the called program. **XQ>XR** is not strictly a "full-page" function, but it only operates on RAM pages thus its inclusion here is justified.



will be shown while the conversion occurs.

- Lastly **CODE** and **DECODE** are the ubiquitous NNN<>HEX functions present in every ROM worth its name – in fact the PowerCL has two sets of them, ONE IN THE "-HEPAXA2" SECTION and another as part of the "-PG FNS" section, aptly named (you'll never guess) **CDE** and **DCD** respectively. We can't have enough of a good thing, or so it seems...

An example to impress your friends: decode the contents of the Status "c" register.



Switch on ALPHA to see the complete contents scroll.



©Photo By Jürgen Keller, 2011.

5.5. Flag Handling functions.-

Modest but still interesting – the functions below round up the flag handling capabilities.

XCF	Extended CF	Allows ANY flag#	Michael Katz
XSF	Extended SF	Allows ANY flag#	Michael Katz
TOGF	Toggle Flag	Allows ANY flag#	Ken Emery
FC?S	Is Flag Clear and Set it	Allows ANY flag#	Ken Emery
FS?S	Is Flag Set and Set it	Allows ANY flag#	Ken Emery

- **XCF** and **XSF** are natural extensions to the mainframe standard Clear/Set Flag functions. Unlike those, the input is expected in X as a decimal entry. Also unlike them they'll accept anyone of the 56 flags from 0 to 55. When used in a program enter the flag# in the preceding program line.
- **TOGF** is a toggle function, inverting the status of the flag which number is in X. It's equivalent to **IF**, the Invert Flag routine in the PPC ROM. See the PPC ROM manual pages 217 and 218 for fun examples altering the status of the system reserved flags.
- **FS?S** and **FS?C** are the symmetric counterparts to the native **FC?S** and **FC?C** functions. They extend the logic and provide shorter handling – sometimes not possible without them. They also operate on the complete flag range, as you'd expect.

Probably not a bad moment for a quick **flag recap**, see the table below:

0-4	shown when set	30	catalog set
5-8	general-purpose	31	date mode: 0)M.DY 1)D.MY
9-10	matrix end of line/column	32	IL man I/O mode
11	auto execution	33	can control IL
12	print double width	34	prevent IL auto address
13	print lower case	35	disable auto start
14	card reader allow overwrite	36-39	number of digits, 0-15
15-16	HPIL printer mode: 0) manual; 1) normal 2) trace; 3)trace w/stack print	40-41	display format: 0) sci; 1) eng 2) fix; 3) fix/eng mode)
17	record incomplete	42-43	angle mode: 0) deg; 1) rad 2) grad; 3) rad
18	IL interrupt enable	44	continuous on
19-20	General-Purpose	45	system data entry
21	printer enabled	46	partial key sequence
22	numeric input available	47	shift key pressed
23	alpha input available	48	alpha keyboard active
24	ignore range errors	49	low battery
25	ignore any errors & clear	50	set when message is displayed
26	audio output is ignored	51	single step mode
27	user mode is active	52	program mode
28	radix mark: 0). 1),	53	IL I/O request
29	digit groupings shown: 0) no; 1) yes	54	set during pause
		55	printer exists

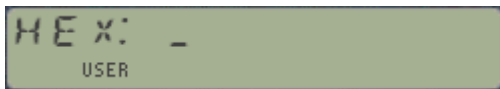
5.6. Other Miscellaneous utilities.

The following functions perform housekeeping tasks and are included in the POWERCL for added convenience.

HEXIN _	HEX Input	1-9, and A-F	Håkan Thörngren
HEXKB _	HEX Entry	1-9, and A-F	Clifford Stern
ROMCAT _ _	ROM Catalog	Starts CAT'2 at ROM id#	J.D. Dodin
SPEED	CPU cycles per second	No input	Doug Wilder
T>BS _ _	Base Ten to Base	Prompts for Base	Ken Emery
PLNG _	Program Length	Program name in prompt	W&W GmbH
RTN?	Pending Returns?	Tests for RTN stack	Doug Wilder
XROM _ _	Rom function Launcher	Prompts for values	Clifford Stern
XQ>GO	XEQ to GO	Drops one RTN Address	Håkan Thörngren
DCD	NNN to HEX string	NNN in X	W&W GmbH
CDE	HEX string to NNN	Hex string in Alpha	Ken Emery

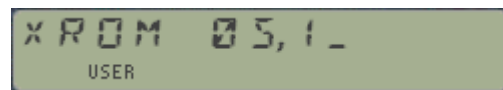
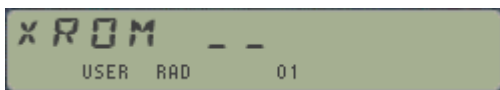
Some brief comments pertaining to each function follow:

- **HEXKB** is the well-known **HEXNTRY** function published in Ken Emery's book "*MCODE for beginners*", and originally written by Clifford Stern. For all purposes it supercedes **CODE** (or **CDE**), which is available in the AMC_OS/X module anyway.
- **HEXIN** does basically the same thing, except that it uses the text in Alpha (if any) as prompt (useful in programs). Use Back-Arrow to delete digits and R/S to terminate the data entry.



You may be wondering how come this is a prompting function, if it is located in a bank-switched page – and the answer is that such is possible as long as the partial key entry method is not employed. This is the case here, and also in **SUMPG** as well – both functions use a key-pressed detection loop as alternative approach, more demanding on power requirements as the CPU doesn't get to Light Sleep - and therefore no switching back to bank-1. The drawback of course is the higher battery consumption, not to be underestimated.

- **XROM** is a well-known function to directly call any function within a plug-in ROM, knowing its ROM id# and function#. Written by Clifford Stern in the heydays of the 41 systems, with a real inside knowledge of the internal OS routines [PARSE]. Both prompt inputs are to be entered as DECIMAL values.



- **ROMCAT** was written by Jean-Daniel Dodin, well-known MCODE pioneer and HP-41 old hand. It prompts for the XROM id# and starts CAT'2 from the position used by such ROM (if present). The usual conventions apply, whereby only the ROM headers are listed unless the catalog is stopped and ENTER^ is pressed in manual mode.

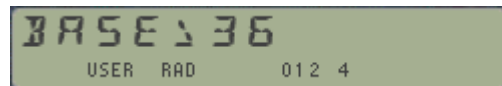


- **SPEED** is a curious gem, although I'm not completely sure I managed to transcribe it well. It's supposed to return the number of CPU cycles per second, so I thought it'd be ideal for the CL given the different TURBO modes. Alas, it always returns the same value (1,126.316), irrespective of the TURBO setting. This is about 6 times bigger than the normal HP-41 result, (167,333) for what is worth. We have Doug Wilder to thank (again) for writing it, using the Time module to keep pace with things.
- **PLNG** asks for the program name in the prompt and returns the program length in bytes. This function is not programmable – and it's extracted from the CCD ROM. Note that the ALPHA mode is turned on automatically for you – no need to press it twice.



- **XQ>GO** is a nifty function that drops one RTN address from the subroutine RTN stack. This can come very handy when you don't want to return to the calling XEQ while running a subroutine, for example depending on the partial results obtained.
- **RTN?** Is related to the same subject: it's another test function that returns YES/NO depending on whether there are pending returns in the subroutine RTN stack. In fact you could use them combined, to cancel the return only if there's a pending RTN to go back to.
- **DCD** and **CDE** are the classic NNN to/from Hex utilities, also used as subroutines throughout the module and thus made available to the user as individual functions as well. Use them to decode the output of **PEEKR**, or to encode the input for **POKER** (albeit **HEXIN** or **HEXKB** provide a more interactive way to do the latter – see next section for details). Note that here too functions **CODE** and **DECODE** in the "-HEPAX2" group basically do the same thing.
- **T>BS** (Ten to Base) is a prompting function for base conversions – where in RUN mode you input the destination base. The result is show in the display and also left in ALPHA. Note that the original argument (decimal value) is left in X unaltered, so you can use **T>BS** repeated times changing the base to see the results in multiple bases without having to re-enter the decimal value.

T>BS is programmable. In PRGM mode the prompt is ignored and the base is expected to be in the Y register. Obviously using zero or one for the base will result in *DATA ERROR*. The prompt can be filled using the two top keys as shortcuts, from 1 to 10 (A-J), or the numeric keys 0-9. The maximum base allowed is 36 – and the "*BASE>36*" error message will be shown if that's exceeded (note that larger bases would require characters beyond "Z").



The maximum decimal value to convert depends on the destination base, since besides the math numeric factors; it's also a function of the Alpha characters available (up to "Z") and the number of them (length) in the display (12). For b=16 the maximum is 9999 E9, or 0x91812D7D600. **T>BS** is an enhanced version of the original function included in Ken Emery's book "MCODE for Beginners". The author added the PRGM-compatible prompting, as well as some display trickery to eliminate the visual noise of the original implementation. Also provision for the case x=0 was added, trivially returning the character "0" for any base.

5.7. Extra Functions – Miscellaneous Con't.

Well yes, there is more! - The table below lists the remaining of miscellaneous functions in the secondary FATs; still excluding the "-HEPAXA2" and "-56BITS" groups.

The underlying idea is to have the most complete function set always available within in those bank-switched pages of the POWERCL – so that there's no need to PLUG other modules, or at least to do it as less frequently as possible. A careful selection of candidates should have screened the less-useful functions, although of course that's always a subjective view.

What follows are the brief descriptions of the functions, grouped them by functionality area - Starting with the system information group.

CRTN?	Curtain location	No inputs	Ken Emery
DREG?	# of Data Registers Set	No inputs	Ken Emery
DSP?	# of Decimal Digits Set	No inputs	Ángel Martin
FREG?	# of Free Registers	No inputs	Ken Emery
ΣRG?	Stat RGs location	No inputs	Ken Emery

- **CRTN?** Returns to X the absolute address of the curtain (i.e. separation between program and data registers). No input value is required. The general equation is: Total Registers = Data Regs + Program Regs; where: TotalRegs = 512 on the CV and CX models.
- **DREG?** Is another SIZE finder, slightly faster than the native version **SIZE?**
- **FREG?** Returns to X the number of available (free) program registers in Main Memory. No input value is required.
- **DSP?** is used to return the number of decimal places currently selected in the display. This is independent from the decimal mode, FIX / SCI / or ENG.
- **ΣRG?** Returns the current location of the Statistical Registers - i.e. those used by the Statistical functions to accumulate the data pairs. Basically it's identical to the CX function **ΣREG?**.

READF	Reads Data File from IL	Disk FileName in ALPHA	R. del Tondo
WRTDF	Writes Data File to Disk	Disk FileName in ALPHA	R. del Tondo
FSIZE	HP-IL Disk File Size	Disk FileName in ALPHA	Dutch PPC members
CVIEW	Non-stop AVIEW	Text in ALPHA	Frits Ferwerda

- **FSIZE** Returns to X the length in registers of the (primary) mass storage file which name is specified in Alpha. If no HP-IL is present on the system an error message will be shown.
- **READF** and **WRTDF** are used to read and write individual DATA files between the IL Drive and XMEM. They are slightly more complicated to use than other file types (like **GETAS** and **SAVEAS** do for ASCII file types), because the Data file must be previously created in the destination. Remember also that **READXM** and **WRTXM** operate on the whole XMEM contents, not on individual files. See [their description](#) paragraphs above.

ΣDGT	Mantissa Digits sum	Number in X	Ángel Martin
VMANT	Views Full mantissa	Number in X	Ken Emery
ST<>Σ	Swaps Stack and StatRG	No inputs	Nelson C. Crowle
X<I>Y	Swaps INDirect X<>Y	No inputs	Nelson C. Crowle

- **ΣDGT** is a divertimento utility that calculates the sum of all mantissa digits, returning it to the X registers. You can use it as a pseudo-random number generator, or just for the sake of it.
- **VMANT** shows the full mantissa of the value in X. The display will present it briefly, and then will return to the standard display settings – unless you press and hold any key during the initial displaying. Good eye-hand coordination is required :-)
- **ST<>Σ** exchanges the contents of the Statistical registers with the Stack. Combined with **ΣREG __** this can be useful to MOVE or SWAP blocks of registers in groups of five, using the stack as intermediate conduit. For instance:

ΣREG_10, ST<>Σ, ΣREG_20, ST<>Σ will move R10-R15 to R20-R25. This is just 6 bytes needed, as opposed to 11 if we were using the **REGMOVE** with its control word in X: 10,020005.

- **X<I>Y** is also about register swapping – this time using X and Y as indirect pointers, so it's equivalent to: RCL IND Y, RCL IND Y, STO IND T, RDN, STO IND Y.

GETPC	Gets Program Counter	No inputs	W&W GmbH
PC<>RTN	Exchanges PC and RTN	No inputs	W&W GmbH
PUTPC	Puts Program Counter	PC location in X	W&W GmbH

- **GETPC, PUTPC, and PC<>RTN** are program-pointer manipulation functions. Use them to recall the location of the current PC, to re-set to another position, or to exchange it with the (last) subroutine return address. They are to be used with a solid understanding of their capabilities (and possible consequences). These functions are not as user-friendly as the CCD Module counterparts **PC>X** and **X>PC**, as proven by the fact that their input and output are NNN. You can use **DCD** to decode the actual addresses.

COMPILE	Compiles jump distances	Global LBL in ALPHA	Frits Ferwerda
TGPRV	Toggle PRIVATE status	Program Name in Alpha	W&W GmbH
CSST	Continuous SST	PC position on program	Phi Trinh
GTEND	Goes To .END.	No inputs	Ken Emery
LASTP	Go to Last Program	No inputs	ZENROM Manual

- **TGPRV** is the inevitable Set/Clear Private status functions – with a twist. To use it the program name must be in ALPHA. This includes programs in RAM or in sRAM (seen as ROM by the calculator). If Alpha is empty the program pointer must set to any line within the program. TGPRV is programmable.
- **GTEND** Sends the program counter to the permanent .END. in program memory (the position of the Curtain). Almost identical to it is **LASTP**, the difference being that the program pointer is placed at the first line of the Last program in RAM, i.e. the one closest to the .END.

- **COMPILE** is a very powerful function that writes all the jump distances in the GOTO and XEQ instructions within the program named in ALPHA. This is extremely useful when uploading a program to a Q-RAM device, like the HEPAX RAM. Having all the jumping distances compiled expedited the execution of the program (no need to search for the label), and also guarantees that short-form GOTO's are not used inappropriately.

- They are feedback messages shown during the execution, indicating which type of instructions are being compiled: 2-Byte GOTO's, and 3-Byte GOTO/XEQ's.



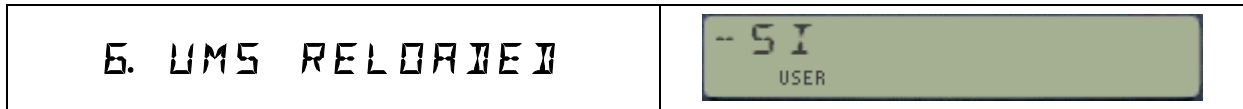
- When the work is done, the message "READY" is shown to inform the user that the execution is completed. Alternatively if a label is missing the execution stops with the program pointer set at the GTO/XEQ statement, and a working message is shown:



- **CSST** Sequentially displays the program steps of the program pointed at by the Program Counter (PC). It's equivalent to using the **SST** key multiple times, and thus its name. The delay between lines shown can be adjusted by pressing any keyboard key, see the original article in PPCJV9N7 p49 for further details. To use it, position first the PC at the target location (using GTO or similar). Note that it won't list PRIVATE programs.

And last (and probably also least for many, not interested in computer science) let's mention the digital bit manipulation functions, as present in the "-56BIT" section of the auxiliary FAT. The table below lists the functions, which operate on the complete 56-bit register as word size (no sophistication here like in the CCD Module).

Function	Description	Author
1CMP	Sets 1-Complement mode	Frits Ferwerda
2CMP	Sets 2-Complement mode	Frits Ferwerda
X+Y	Bitwise addition of values stored in X and Y; Result left in X	Gordon Pegue
Y-X	Bitwise subtraction of values in X and Y, Result left in X	Gordon Pegue
XANDY	Logical AND of values in X and Y, Result left in X	Gordon Pegue
XORY	Logical OR of values in X and Y, Result left in X	Gordon Pegue
NOTX	Logical NOT of values in X and Y, Result left in X	Gordon Pegue
RXR	Rotate right 56-bit field in X one digit (4 bits)	Gordon Pegue
RXL	Rotate Left 56-bit field in X one digit (4 bits)	Gordon Pegue
BRXL	Rotate Left 56-bit field in X one bit w/ wraparound	Gordon Pegue
RLN	Rotate Left 56-bit field in Y N digits, w/ N is in X	Gordon Pegue
RRN	Rotate Right 56-bit field in Y N digits, w/ N in X	Gordon Pegue



6.1. Unit Management System – A full swing relapse.

Bank-switched modules are a fantastic invention, but there are a few system design limitations that cannot be overcome, not even using this advanced technique. Such is the 64-entries limit in the FAT, one of the absolute barriers (or boundary conditions if you prefer) that exerts its controlling power regardless of the number of shadowed-banks we care to set-up.

It is because of this that bank switching lends itself very well to large bodies of code with few FAT entries, as opposed to many functions of reduced size. The best example is the HEPAX module, where dedicated banks are arranged for gigantic-code functions – such as **HEXEDIT** and **DISASM**.

In the POWERCL case the triggering point was the ROM table holding the image library information, but even being large enough (over 1k) it doesn't fill up the second bank completely. The idea of adding the Unit Management System came as a natural to "fill the gap", because it too has the ideal attributes to be placed on the second bank: huge chunks of code, used sequentially and independently from the rest of the module, with tables and hard-coded values to be read.

The idea was very appealing; as it removes the need to have the Unit Conversion module loaded and saves one block for other uses – two big scores in one. It also provided the opportunity to write a Constants Library *a la* 33S or 35S, sorely missing even in the UMS module.

So chances are you'll never use it, but even then because they're all tucked away in a secondary bank it's not adding any burden to your system configuration, or compromising the available resources. And if you use it, well I think you'll be pleased with the end result. Can anyone ask for more?

-SI	Direct Unit Conversion	FROM-TO in Alpha	HP Co. / Á. Martin
A<>RG	Swaps Alpha and Regs	Alpha <> Regs	Á. Martin
KLIB	Constant Library	4 line-ups w/ 5 each	Á. Martin
SI-	Inverse Unit Conversion	TO-FROM in Alpha	HP Co. / Á. Martin
ST<>RG	Swaps Registers	Stack <> Reg	Á. Martin
UCAT	UNIT Catalog	Prompts for Type	P. Platzer / A. Martin

The main conversion functions are obviously **-SI** and **SI-**, where the hyphen sign indicates the direction of the conversion "to/from" the International System of units (*Systeme Internationale* in French); or in mixed specifications in the from/to unit string.

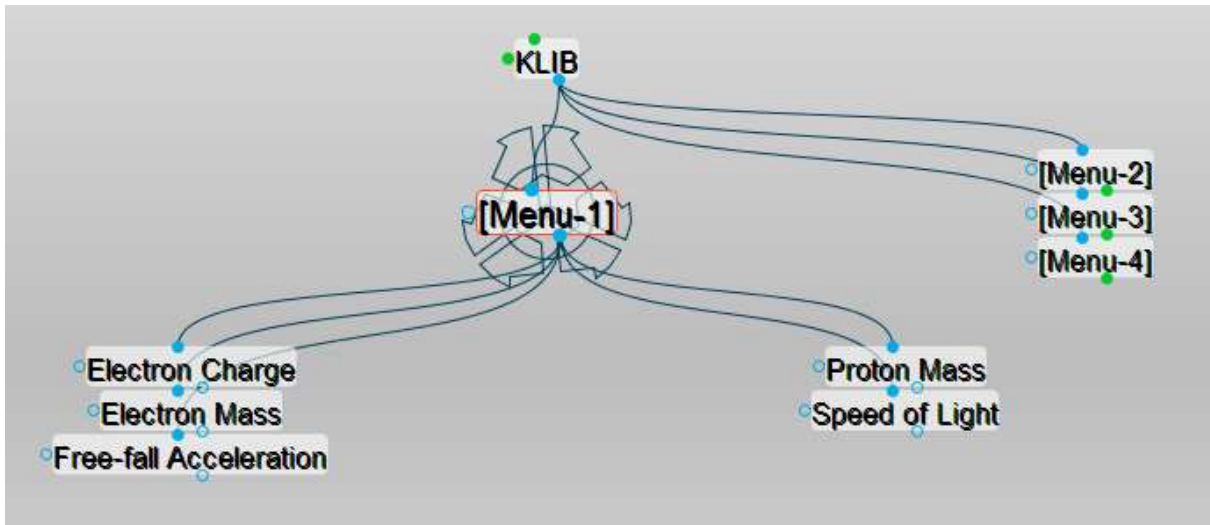
The auxiliary functions **A<>RG** and **ST<>RG** as added for convenience.- Use then to save complete ALPHA unit strings in consecutive data registers, as well as making temporary back-ups of the stack contents. They are both prompting functions in RUN mode, asking for the initial data register.

Functions have been slightly modified to comply with the bank-2 requirements, and **UCAT** is also enhanced with a new unit type prompt ("**G:F:T:W:E**" for Geometry, Force, Temp/Mass, Work, and Electrical units) and better internal structure.

6.2. An enhanced Constants Library.

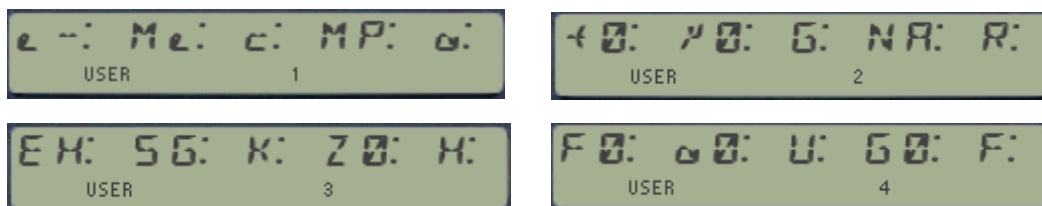
KLIB is the new addition to the Unit Conversion. In its original implementation the module included 15 constants, each of them setup as an individual FAT entry. This allowed individual access in RUN and PRGM modes, but however lacked a library catalog - offering a more convenient way to access them.

Besides the obvious advantage in usability, with this new implementation only one FAT entry is used – allowing for more functions to be added to the ROM. *The use in PRGM mode is also possible by selecting the specific constant by its index* (a number from 1 to 20), as independent program line following **KLIB** (thus the same non-merged approach also used elsewhere).



The following improvements have been made:

- Added 5 more constants to the library – for a total of 20. Not surprisingly quite a few of them are related to the EE field, showcasing both the UMS extensions and the author's background.
- Constants are presented in the display as line-up menus, with 4 groups of 5 entries each.
- In RUN mode, the selection is made using the top row keys, with the SST key used to navigate between the four "line-up" screens. Flag annunciators 1-4 indicate the current screen, so you know where you are.
- In PRGM mode the selection is made by adding an index number after **KLIB**, using the non-merged functions technique: two program lines in total for the complete information, not disrupting the stack. Note that the function will not offer selection screens in this case.
- The constant value is placed in the X register (stack is lifted), and its units are written in Alpha – ready for any unit conversion activity.



The figure above shows the four line-ups available in **KLIB**, to select the constant from the library. Note the flag annunciator indicating the one currently selected.

Because of the limited capabilities of the HP41 display, some of the symbols used to represent the constants differ from the standard notation. Slightly improved notation is possible using the lower case letters available in the "halfnut" character set but I have used the standard "fullnut" character set for compatibility with all 41 models (and the CL specially).

KLIB follows the same model available in newer calculators, like the 33S and the 35S, so chances are that you're probably already familiar with it. Use the SST key to navigate to the next menu screen, from 1 to 4 repeated in sequential way (no backwards choice). The appropriate flag indicator – 1,2,3,4 – will tell the current "screen" shown, so you'll know where you are. Use the BackArrow key to cancel out – and no constant will be selected.

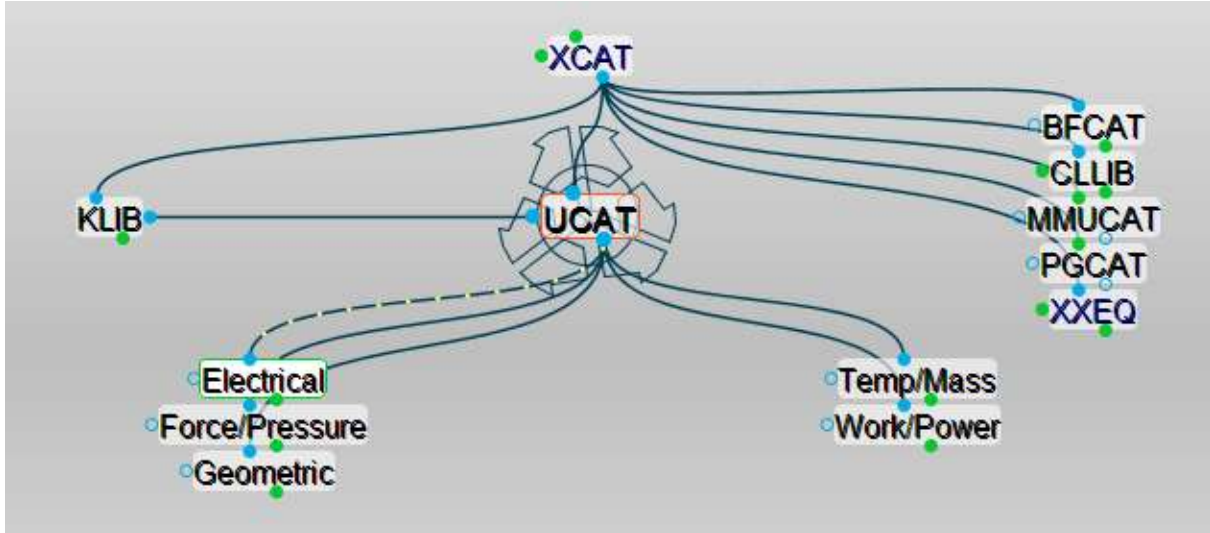
There is no visual feedback when you choose the constant using keys A-E. its value and units will be directly entered in the X and ALPHA registers – or the action cancelled (showing 'NULL') if you hold the key depressed for a while.

The constants included, their values and appropriate units in which they are expressed are listed in the following table (note the first column with the index for PRGM operation):

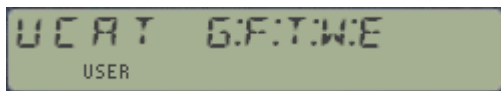
#	Name	Description	Value	Units
1	e-	Electron Charge	-1,6021764 E-19	CB
2	Me	Electron Mass	9,1093821 E-31	KG
3	c	Speed of Light	2,9979245 E 08	M/S
4	MP	Proton's Mass	1,6726216 E-27	KG
5	a	Free-fall Acceleration	9,8066500 E 00	M/S2
6	ε0	Vacuum Permittivity	8,8541878 E-12	FD/M
7	μ0	Vacuum Permeability	1,2566370 E-06	N/A2
8	G	Gravity Constant	6,6742800 E-11	N*M2/KG
9	NA	Avogadro's Number	6,0221417 E 23	1/MOL
10	R	Gas Constant	8,314472150	J/K*MOL
11	EH	Hartree Energy	4,359748226 E-18	J
12	SG	Stefan-Boltzmann	5,6705119 E-8	W/M2*K4
13	K	Boltzmann's Constant	1,3806504 E-23	J/K
14	Z0	Vacuum impedance	376,7303134	OHM
15	H	Planck's constant	6,6260689 E-34	J*S
16	FO	Magnetic flux quantum	2,0678336-15	WB
17	a0	Bohr Radius	5,2917720-11	M
18	U	Atomic mass unit	1,6605387-27	KG
19	GO	Conductance quantum	77,480916-06	1/OHM
20	F	Faraday constant	96.485,34	CB/MOL

5.3.- Unit Conversion Catalog.

As the unit tables get larger it becomes more challenging to remember the exact spelling of each unit symbol, resulting in frequent Invalid Conversion error conditions. A Unit Catalog is therefore almost an obvious addition to the UMS – and as such it was mentioned in the old HP documentation as a next addition, which obviously never came into being. Until now, that is.



UCAT lists all unit symbols sequentially, following the order in which they are stored on the Unit Table. This used to be alphabetical in the original implementation – but has been modified in this new incarnation for reasons explained in the next paragraph. **UCAT** adds a prompting entry to select the *magnitude section* to start the listing from. The section prompts are as follows:



- G: Geometry Section (length, surface and volume units)
- F: Pressure and Force Section
- T: Matter and Mass section (Mass, Density, Viscosity, Temperature)
- W: Energy, Power and Time Section
- E: Electrical and Luminance Section

The units listing will continue until the end of the table is reached (i.e. end of section 5) regardless of where it got started. This is not perfect but good enough for the majority of circumstances where one's looking for the appropriate specific unit symbol.

Navigating the Unit Catalog.

The catalog can be paused and resumed at any time using R/S. Besides, you can use SST and BST to single-step the units forwards and backwards, and resuming the listing with SHIFT activated will list them in reverse order. Use the arrow key to stop the listing and return to the main OS.

- A running catalog will automatically terminate when it reaches the end of the unit table (or the beginning if running backwards).
- A single-stepped catalog will not go beyond the last unit (or first one if moving backwards) even if you keep pressing SST (or BST). To terminate it you can press R/S or the back arrow key.

- Use R/S to toggle between continuous and single-step catalog display at any time during the execution.

Catalog Hot-keys.

With the unit catalog paused, (i.e. in single-step mode) you can use the following keys to **directly edit the unit string in the Alpha register** in the following manner:

- **[ENTER]** – Clears Alpha and adds the displayed unit symbol to the string.
- **[SHIFT],[ENTER]** – Appends the displayed unit as *destination field* in the unit string - i.e. appends "-" plus the unit symbol to the text already existing in Alpha.
- **[*]** – Appends the displayed unit as multiplying unit (i.e. appends "*" plus symbol)
- **[/]** – Appends the displayed unit as dividing unit (i.e. appends "/" plus symbol)

In this way it's rather simple to build complete unit string just by pressing the corresponding hot keys during the catalog listing – no need to remember the exact spelling of the unit symbols. Obviously it's still the user's task to recognize the symbol and identify it with the corresponding unit name. Of course you can edit the Alpha register directly as always just typing the syntax in Alpha mode - if that's your preferred choice.

Note that there are no checks for the string built – so it's possible to press **[/]** multiple times, or repeat **[SHIFT],[ENTER]** – which obviously would not be a valid string. Note also that after every usage of a hot key you need to re-start the catalog listing again if you want to continue to build the complete unit string. A small price to pay for the convenience to occasional users (aren't we all?) to avoid syntax errors!

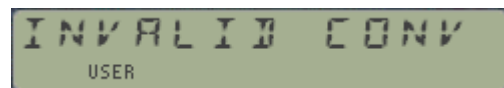
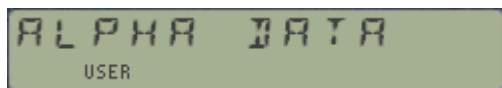
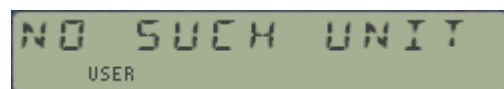
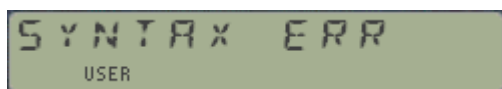
Meaningful Error Conditions.

The Thermal Pac only used standard OS error messages (like **ALPHA DATA** and **DATA ERROR**) to inform the user of an error condition. It did NOT report unit spelling errors either, ignoring them even if user flag 25 was cleared! The Petroleum Pac improved on that with its dedicated error message "Invalid Conversion".

This however wasn't very informative, as it didn't indicate where exactly was the problem: either a syntax error (like typing the sigma or percent sign), an invalid unit string (non-homogeneous source and destinations), or a misspelled unit symbol all produced the same "**INVALID CONV**". Besides it didn't signal the ALPHA DATA condition anymore – clearly a step back here.

The Unit Conversion Module recovers the alpha data message and adds two new messages to the Invalid Conversion condition – offering four error-trapping cases, and so making error detection and correction a much easier task, needless to say. The messages are as follows:

- ALPHA DATA – when a non-numeric input is in X
- SYNTAX ERR – when using illegal characters (like lower-case letters)
- NO SUCH UNIT – when the alpha string contains a symbol not on the unit table
- INVALID CONV – when the unit string is not dimensionally correct.



Example:- Car Consumption comparison (Gas/Petrol)

The short routine below converts between the two (mystifying) conventions used in the US and Europe to rate the gas consumption of a car:

US: Milles per Gallon, vs
 Europe: Liters per 100 km.

And here's a little table compiled with it:

Mi/Gal	L/100km
15	15.681
20	11.761
25	9.409
30	7.840
35	6.720

1	LBL "->US"
2	XEQ 00
3	-SI
4	E2
5	/
6	1/X
7	RTN
8	LBL "->EU"
9	1/X
10	E2
11	*
12	XEQ 00
13	SI-
14	RTN
15	LBL 00
16	"L/KM-GAL/MI"
17	END

Appendix.- Module Unit Conversion Comparisons.

The following five sections offer a detailed comparison between the implementations of the Unit Conversion System, as found in the different modules.

1. Geometry Units.				UnitCon	Petroleum	Thermal	Machine
Symbol	Unit	Magnitude	Group				
ACRE	acre	Surface	Geometry	1	1	-	-
ANG	angstrom	Length	Geometry	1	-	1	1
AU	Astronomical Unit	Length	Geometry	1	-	-	-
BBL	Barrel of petroleum	Volume	Geometry	1	1	1	-
CM	Centimeter	Length	Geometry	1	1	1	1
FT	Foot	Length	Geometry	1	1	1	1
GAL	Gallon (US)	Volume	Geometry	1	1	1	1
GALUK	Gallon (UK)	Volume	Geometry	-	1	-	-
IN	Inch	Length	Geometry	1	1	1	1
KM	kilometer	Length	Geometry	1	1	1	1
L	liter	Volume	Geometry	1	1	1	1
LY	Light Year	Length	Geometry	1	-	-	-
M	meter	Length	Geometry	1	1	1	1
MI	mile	Length	Geometry	1	1	1	1
MIC	micron	Length	Geometry	1	-	1	1
MIL	1/1000 inch	Length	Geometry	1	-	1	1
ML	milliliter	Volume	Geometry	1	1	1	1
MM	millimeter	Length	Geometry	1	1	1	1
PC	Parsec	Length	Geometry	1	-	-	-
UM	micrometer	Length	Geometry	-	1	-	-
YD	yard	Length	Geometry	1	1	1	1

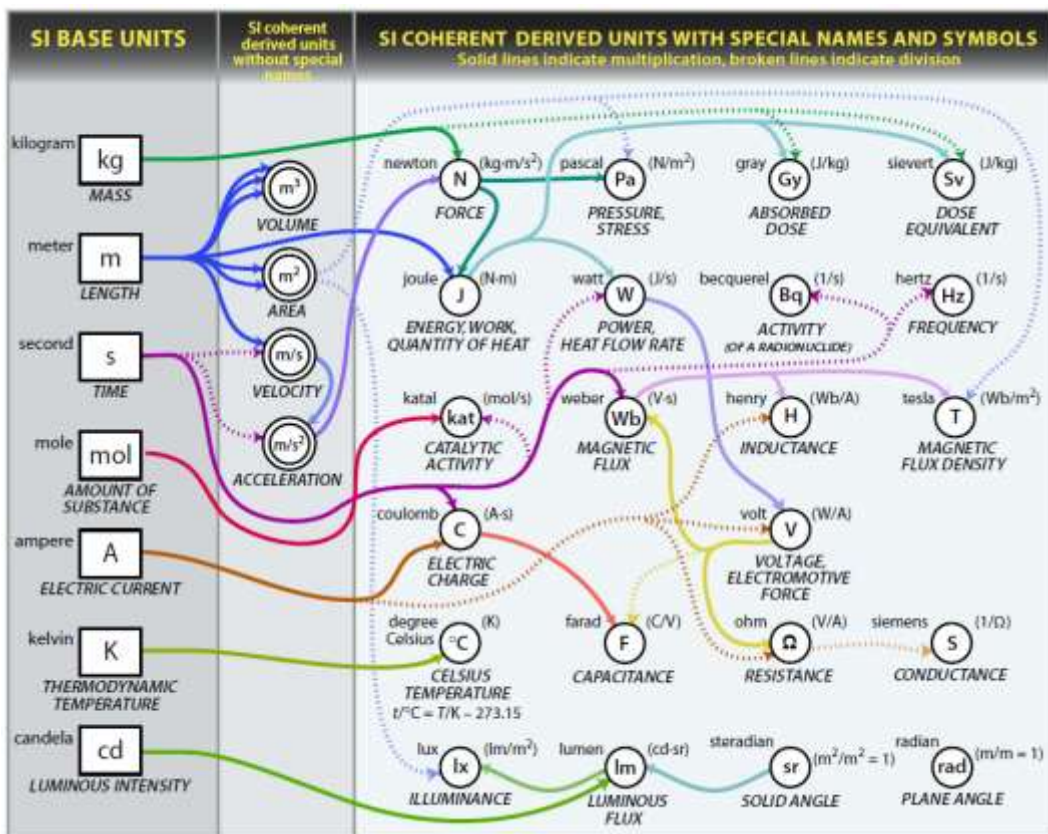
2. Force and Pressure Units.				UnitCon	Petroleum	Thermal	Machine
Symbol	Unit	Magnitude	Group				
ATM	Atmosphere	Pressure	Force & Pressure	1	1	1	1
BAR	Bar	Pressure	Force & Pressure	1	1	1	-
DYNE	Dyne	Force	Force & Pressure	1	1	1	1
FTH2O	Foot of Water	Pressure	Force & Pressure	1	1	1	-
INHG	Inch of Mercury	Pressure	Force & Pressure	1	1	1	-
INH2O	Inch of Water	Pressure	Force & Pressure	1	1	1	-
KGF	Kilogram Force	Force	Force & Pressure	1	1	1	-
KIP	Kilopound Force	Force	Force & Pressure	1	1	1	-
KPA	kilopascal	Pressure	Force & Pressure	1	1	1	1
KSI	KIP per square inch	Pressure	Force & Pressure	-	1	-	-
LBF	pound force	Force	Force & Pressure	1	1	1	1
MBAR	milibar	Pressure	Force & Pressure	-	1	-	-
MMHG	millimeter of mercury	Pressure	Force & Pressure	1	1	-	-
MN	meganewton	Force	Force & Pressure	-	1	-	-
MPA	megapascal	Pressure	Force & Pressure	1	1	-	-
N	newton	Force	Force & Pressure	1	1	1	1
PA	pascal	Pressure	Force & Pressure	1	1	1	1
PDL	poundal	Force	Force & Pressure	1	-	1	1
PSF	pound force per square foot	Pressure	Force & Pressure	1	1	1	1
PSI	pound force per square inch	Pressure	Force & Pressure	1	1	1	1
TORR	torr	Pressure	Force & Pressure	1	1	1	-

3. Matter & Mass Units.				UnitCon	Petroleum	Thermal	Machine
Symbc	Unit	Magnitude	Group				
API	Degree API	Density	Matter & Mass	-	1	-	-
BCF	Billion Cubic Feet of	Gas Volume	Matter & Mass	-	1	-	-
C	Degree Celsius	Temperature	Matter & Mass	1	1	1	1
CP	Centipoise	Viscosity	Matter & Mass	-	1	-	-
CST	Centistoke	Kinematic Viscosit	Matter & Mass	-	1	-	-
D	Darcy	Porosity	Matter & Mass	1	1	-	-
DA	Dalton	Mass	Matter & Mass	1	-	-	-
F	Degree Farenheit	Temperature	Matter & Mass	1	1	1	1
G	Gram	Mass	Matter & Mass	1	1	1	1
K	Kelvin	Temperature	Matter & Mass	1	1	1	1
KG	Kilogram	Mass	Matter & Mass	1	1	1	1
KMOL	kilomole	Matter	Matter & Mass	-	1	-	-
KT	kilotonne	Mass	Matter & Mass	-	1	-	-
LBM	pound mass	Mass	Matter & Mass	1	1	1	1
MCF	thousand cubit feet	Gas Volume	Matter & Mass	-	1	-	-
MD	millidarcy	Porosity	Matter & Mass	-	1	-	-
MG	megagram	Mass	Matter & Mass	-	1	-	-
MMCF	million cubic feet ga	Gas Volume	Matter & Mass	-	1	-	-
MOL(E)	mole	Matter	Matter & Mass	1	1	1	-
MT	megatonne	Mass	Matter & Mass	-	1	-	-
P(OISE)	poise	Viscosity	Matter & Mass	1	1	1	-
R	degree rankine	Temperature	Matter & Mass	1	1	1	1
SCF	standard cubic foot	Gas Volume	Matter & Mass	-	1	-	-
SCM	standard cubic mete	Gas Volume	Matter & Mass	-	1	-	-
SCMZ	standard cubic mete	Gas Volume	Matter & Mass	-	1	-	-
SLUG	slug	Mass	Matter & Mass	1	-	1	1
SPGR	specific gravity to wa	Density	Matter & Mass	-	1	-	-
ST(OKE)	stoke	Kinematic Viscosit	Matter & Mass	1	1	1	-
T	tonne	Mass	Matter & Mass	-	1	-	-
TON	short ton	Mass	Matter & Mass	1	1	1	1
TONUK	long ton	Mass	Matter & Mass	-	1	-	-

4. Energy, Power & Time Units				UnitCon	Petroleum	Thermal	Machine
Symbc	Unit	Magnitude	Group				
BTU	British Thermal Unit	Energy	Energy & Time	1	1	1	1
CAL	Calorie	Energy	Energy & Time	1	1	1	1
CV	Cheval Vapeur	Power	Energy & Time	1	-	-	-
DAY	Day	Time	Energy & Time	1	1	1	-
ERG	Erg	Energy	Energy & Time	1	1	1	1
EV	Electron-Volt	Energy	Energy & Time	1	-	-	-
HP	Horsepower	Power	Energy & Time	1	1	1	1
HR	Hour (mean solar)	Time	Energy & Time	1	1	1	1
J	Joule	Energy	Energy & Time	1	1	1	1
KCAL	kilocalorie	Energy	Energy & Time	1	1	1	-
KJ	Kilojoule	Energy	Energy & Time	-	1	-	-
KW	kilowatt	Power	Energy & Time	1	1	1	1
MIN	minute	Time	Energy & Time	1	1	1	1
MJ	megajoule	Energy	Energy & Time	-	1	-	-
MO	month	Time	Energy & Time	-	1	-	-
MW	megawatt	Power	Energy & Time	-	1	-	-
S	second	Time	Energy & Time	1	1	1	1
THERM	10^5 BTU	Energy	Energy & Time	-	1	-	-
W	watt	Power	Energy & Time	1	1	1	1
YR	year	Time	Energy & Time	1	1	-	-

5. Electrical & Luminance Units

Symbol	Unit	Magnitude	Group	UnitCon	Petroleum	Thermal	Machine
A	Ampere	Electrical Current	Electrical	1	-	-	-
CB	Coulomb	Electric Charge	Electrical	1	-	-	-
CD	Candela	Luminous Intensity	Electrical	1	-	-	-
FD	Farad	Capacitance	Electrical	1	-	-	-
FDY	Faraday	Electric Charge	Electrical	1	-	-	-
FRK	Franklin	Electric Charge	Electrical	1	-	-	-
GSS	Gauss	Magnetic Field	Electrical	1	-	-	-
HY	Henry	Inductance	Electrical	1	-	-	-
KO	kOhm	Resistance	Electrical	1	-	-	-
LM	Lumen	Luminous Flux	Electrical	1	-	-	-
LX	Lux	Illuminance	Electrical	1	-	-	-
MHY	mHenry	Inductance	Electrical	1	-	-	-
MX	Maxwell	Magnetic Flux	Electrical	1	-	-	-
O	Ohm	Resistance	Electrical	1	-	-	-
OED	Oersted	Magnetic Intensity	Electrical	1	-	-	-
PFD	pFarad	Capacitance	Electrical	1	-	-	-
TS	Tesla	Magnetic Field	Electrical	1	-	-	-
V	Volt	Voltage	Electrical	1	-	-	-
WB	Weber	Magnetic Flux	Electrical	1	-	-	-



Farewell.

And with this you've reached the end of the POWERCL_EXT manual. – I hope these few pages have proven useful to you in your quest to become familiar with its capabilities and what your appetite for even more to come.

The 41CL is an innovative development nothing short of incredible, with amazing possibilities that open the door to yet new tricks on the HP-41 platform; all this still happening 33+ years after the original 41 was launched. *Now that's what I call an achievement!*

Before and after: 33 years separate these boards:

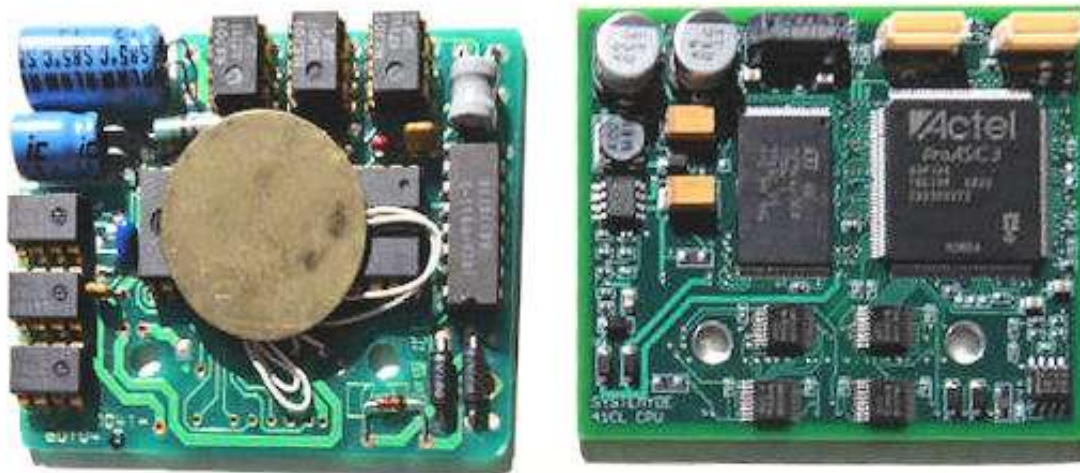
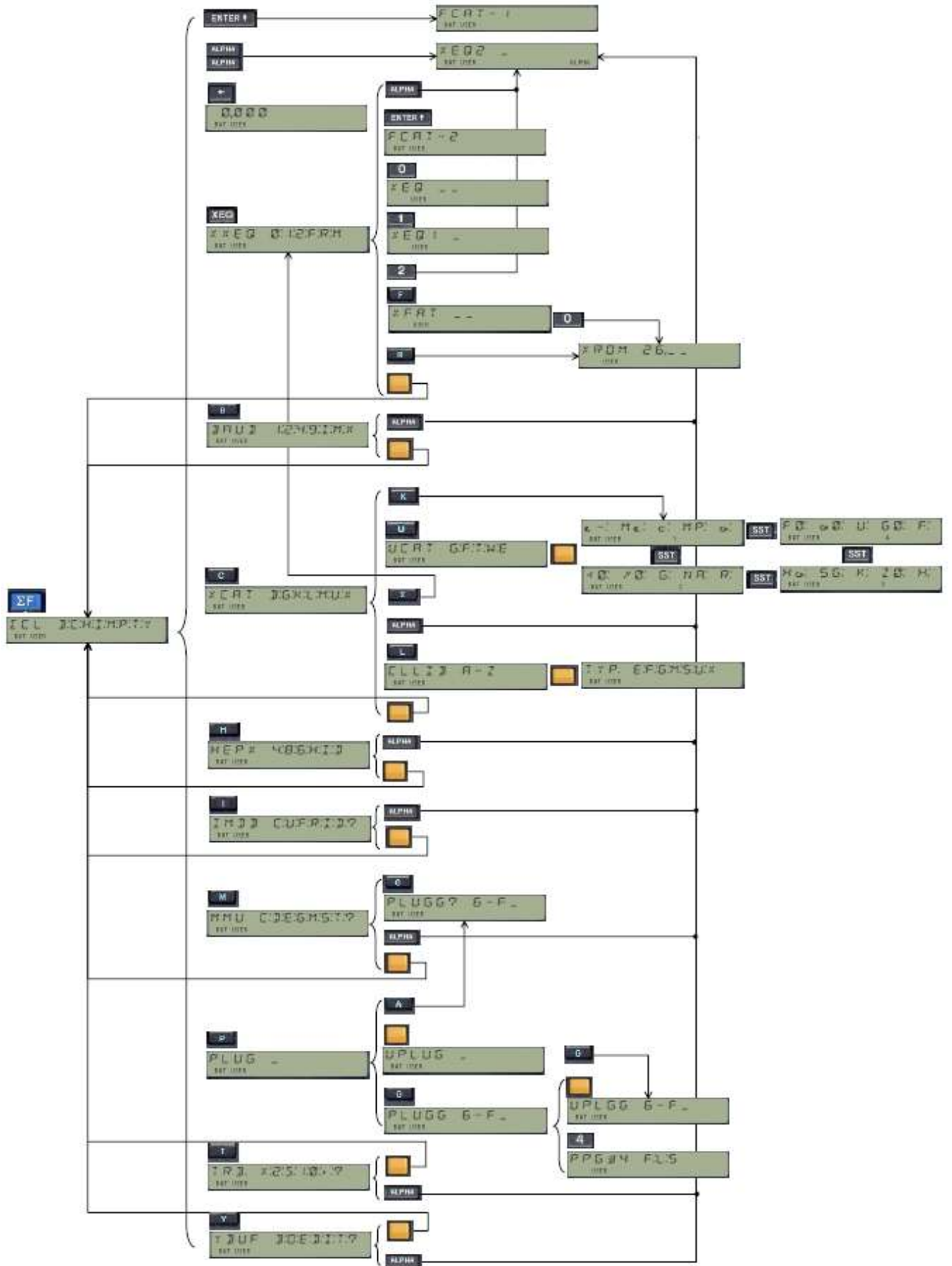


Photo: Steve Leibson, (c) May 2011

Appendices.



Appendix 0. Summary of [ΣCL] Function launching functionality.

There are several function launchers in the POWERCL. We've also seen that they show very flexible and interconnected combinations, sometimes "warping" around the complete module in a lot of ways. It's initially a bit confusing, yet you should not get intimidated by the multiple layers - as you'll get very quickly acquainted to the design and comfortable using it.

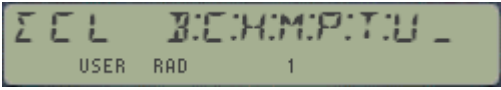
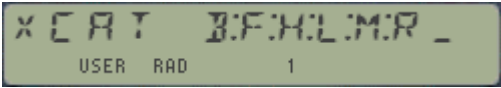
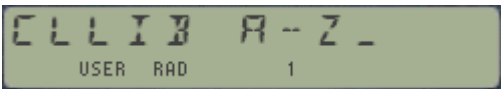
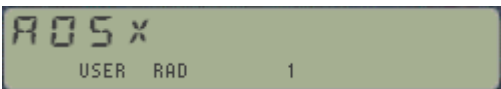

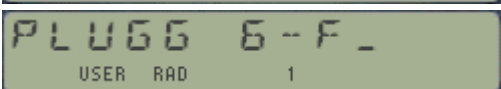
Notice that in some instances the same function can be accessed from two different places (i.e. HEPDIR, MMUCAT). Note also that they may be required to sequentially input the prompts from two (or more) launchers to finally get to the desired end result. –

For instance, say you want to plug the **AMC"OS/X** ROM into port 7 (the HP-IL one). Say you don't remember its CL ROMid#, nor the function name to accomplish that (which happens to be **PLUGH**). You decide to start at the very beginning, executing **ΣCL**, then chose "**C**" for CATALOGS, then "**L**" for CLLIB, then "**A**" to start the enumeration from the letter A, then **R/S** at the "AOSX", followed by "**P**" to invoke the PLUG action. Then "**G**" to go to the page settings, then "**7**". *A breeze !* ☺

Putting it all together now.-

Granted this is one of the more complex examples (but trust me, not the worst one!) Here it is again with more details so it's easier to follow:- Let the display be our guide in this dialog with our trusty companion...

Start off by executing **ΣFL**, which will present:

	, enter " C " for the Catalog Launcher [XCAT]
	, enter " L " for the CL Library [CLLIB]
	, input " A ", R/S , SST to navigate until you see:
	, enter " P " to trigger the PLUG functionality [PLUG]
	, enter " G " to invoke the page-plugging [PLUGG]
	, enter " 7 " - the page number to complete the job.

Easy does it – and you didn't have to remember any mnemonic or function name; nor does it require any key assignment other than **ΣCL** - to call up to 58 different functions. Does it get any better than this?

As a handy summary see the menu map in the left, and the table in next page showing all possible combinations to access a given function, using the different function launchers. Cells show the keys to press to access to the function, starting from the launcher on the column header.

BLACK fns. YFNS Functions

BLUE fns. POWERCL Functions

#	F. Name	ΣCL	BAUD	XCAT	HEPX	MMU	TURBO	PLUG	UPLUG
1	BAUD12	B,1	1						
2	BAUD24	B,2	2						
3	BAUD48	B,4	4						
4	BAUD96	B,9	9						
5	BFCAT	C,F		F					
6	BLCAT	C,B		B					
7	CLLIB _	C,L		L					
8	HEPDIR	H,D		H	D				
9	HEPINI _ "	H,I			I				
10	HPX16 _	H,6			6				
11	HPX4 _	H,4			4				
12	HPX8 _	H,8			8				
13	KLIB _	C,K		K					
14	MMU?	M,?				?			
15	MMUCAT	M,T		M		T			
16	MMUCLR	M,C				C			
17	MMUDIS	M,D				D			
18	MMUEN	M,E				E			
19	PLUG1	P,1		L*,P,1				1	[],1
20	PLUG1L	P,L,1		L*,P,L,1				L,1	[],L,1
21	PLUG1U	P,U,1		L*,P,U,1				U,1	[],U,1
22	PLUG2	P,2		L*,P,2				2	[],2
23	PLUG2L	P,L,2		L*,P,L,2				L,2	[],L,2
24	PLUG2U	P,U,2		L*,P,U,2				U,2	[],U,2
25	PLUG3	P,3		L*,P,3				3	[],3
26	PLUG3L	P,L,3		L*,P,L,3				L,3	[],L,3
27	PLUG3U	P,U,3		L*,P,U,3				U,3	[],U,3
28	PLUG4	P,4		L*,P,4				4	[],4
29	PLUG4L	P,L,4		L*,P,L,4				L,4	[],L,4
30	PLUG4U	P,U,4		L*,P,U,4				U,4	[],U,4
31	PLUGG _	P,G		L*,P,G				G	[],G
32	PLUGG? _	P,A		L*,P,A		G		A	[],A
33	PLUGH	P,H		L*,P,H				H	[],H
34	PLUGP	P,P		L*,P,P				P	[],P
35	PPG#4	P,G,4		L*,P,G,4				G,4	[],G,4
36	ROMLIB _	C,L,[]		L,[]					
37	SERINI	B,I	I						
38	TURBO?	T,?					?		
39	TURBO10	T,1					1		
40	TURBO2	T,2					2		
41	TURBO20	T,0					0		
42	TURBO5	T,5					5		
43	TURBO50	T,""					""		
44	TURBOX	T,X					X		
45	UCAT _	C,U							
46	UPLGG _	U,G		L*,P,[]G				[],1	G
47	UPLUG1	U,1						[],L,1	1
48	UPLUG1L	U,L,1						[],U,1	L,1
49	UPLUG1U	U,U,1						[],2	U,1
50	UPLUG2	U,2						[],L,2	2
51	UPLUG2L	U,L,2						[],U,2	L,2
52	UPLUG2U	U,U,2						[],3	U,2
53	UPLUG3	U,3						[],L,3	3
54	UPLUG3L	U,L,3						[],U,3	L,3
55	UPLUG3U	U,U,3						[],4	U,3
56	UPLUG4	U,4						[],L,4	4
57	UPLUG4L	U,L,4						[],U,4	L,4
58	UPLUG4U	U,U,4						[],G	U,4
59	UPLUGA	U,A						[],A	A
60	UPLUGH	U,H						[],H	H
61	UPLUGP	U,P						[],P	P
62	UPPG4	U,G,4						[],G,4	G,4

Appendix 1 – Detailed ROM id# table – in alphabetical order.

The table below lists all modules included in the CL Library, sorted alphabetically by CL-ID mnemonic.

#	ID	Size	Name	Type	Author
1	441Z	8k	Lib#4 based 41Z	Math	Ángel Martin
2	4ALP	4k	Alpha_44	Utilities	Ángel Martin
3	4AOX	4k	AMC_OS/X 4L	Extensions	Ángel Martin
4	4DIG	4k	Lib#4 41Z Diagnostics	Math	Ángel Martin
5	4LIB	4k	Library#4	Extensions	Ángel Martin
6	4MTR	4k	Lib#4 Matrix ROM	Math	Ángel Martin
7	4PLY	4k	Lib4#4 Polynomial ROM	Math	Ángel Martin
8	4RAM	4k	RamPage_4L	Utilities	Ángel Martin
9	4SM4	16k	Lib4# SandMath 2x2	Math	Ángel Martin
10	4SMT	8k	SandMath_44	Math	Ángel Martin
11	4TBX	4k	ToolBox_4L	Utilities	Ángel Martin
12	4UTL	4k	CL_Utils 4H	Extensions	Ángel Martin
13	A41P	12k	Advantage Pac	Math	HP Co.
14	AADV	4k	Advantage Applications	Science	J-F Garnier
15	ADV1	16k	Adventure_1	Games	Ángel Martin
16	ADV2	12k	Adventure_2	Games	Ángel Martin
17	AEC3	8K	AECROM 13-digit	Engineering	Ángel Martin
18	AECR	8k	AECROM	Engineering	Red Shift
19	AFDE	8k	AFDC1	Engineering	GunZen
20	AFDF	8k	AFDC2	Engineering	GunZen
21	AFIN	4k	Auto Finance	Financial	GMAC
22	ALGG	8k	Algebra ROM	Math	Ángel Martin
23	ALGY	4k	Astro*ROM	Science	Elgin Knowles & Senne
24	ALPH	4k	ALPHA ROM	Utilities	Á. Martin & D. Wilder
25	ANTS	4k	Antennas	Engineering	HP Co.
26	AOSX	4k	AMC OS/X	Engineering	Ángel Martin
27	ASM4	4k	Assembler4	Extensions	??
28	ASMB	4k	Assembler3	Extensions	??
29	ASTT	16k	ASTRO-2010 Module	Science	Jean-Marc Baillard
30	AUTO	4k	Auto-Start / Dupl ROM	Extensions	HP Co.
31	AV1Q	4k	AV1 ROM	Engineering	Beechcraft
32	AVIA	4k	Aviation Pac	Engineering	HP Co.
33	B52B	8k	B-52 ROM	Engineering	Boeing
34	BCMW	4k	BCMW ROM	Engineering	??
35	BESL	8k	Bessel ROM	Math	Á. Martin & JM Baillard
36	BLDR	8k	BLD ROM	Utilities	W. Doug Wilder
37	BLND	4k	Bufferland ROM	Utilities	Ángel Martin
38	BSMS	4k	Bus Sales/Mkt/Stat.	Financial	HP Co.
39	BUD2	8k	Buderus-2	Engineering	WMK
40	BUD3	4k	Buderus-3	Engineering	WMK
41	CCDA	4k	CCD AdvApps.	Utilities	Ángel Martin
42	CCDP	8k	CCD Plus	Utilities	W&W + Angel Martin
43	CCDR	8k	CCD Module	Utilities	W&W
44	CCDX	4k	CCD OS/X	Extensions	W&W + R. del Tondo
45	CENG	4k	Chemical Engineering	Engineering	HP Co.
46	CHEM	4k	Chemistry User ROM	Science	Les Wright
47	CHES	8k	Chess/Rubik's ROM	Games	C. Roetlgen / J. Perry
48	CIRC	4K	Circuit Analysis Pac	Engineering	HP Co.

#	ID	Size	Name	Type	Author
49	CLIN	4K	Clinical Lab Pac	Engineering	HP Co.
50	CLUT	4k	CL Utilities	Utilities	Ángel Martin
51	CMT1	16k	CMT-10 Eprom Test	Extensions	CMT
52	CMT2	4k	CMT-200 ROM	Extensions	CMT
53	CMT3	8k	CMT-300 ROM	Extensions	CMT
54	CNTL	4k	Control Systems	Engineering	HP Co.
55	CURV	8k	Curve-Fitting Module	Math	Ángel Martin
56	CVPK	8k	Cv-Pack ROM	Engineering	??
57	DA4C	4k	DisAssembler 4C	Extensions	W. Doug Wilder
58	DACQ	8k	Data Acquisition Pac	Extensions	HP Co.
59	DASM	4k	DisAssembler 4D	Extensions	W. Doug Wilder
60	DAVA	4K	David Assembler 2C	Extensions	David van Leeuwen
61	DEMO	16k	41 System Demo Module	Extensions	HP Co.
62	DEV2	4k	HP-IL Devil-2	Extensions	HP Co. + ÁM
63	DEVI	8k	HP-IL Development	Extensions	HP Co.
64	DIFF	8k	DIFFEQ ROM	Math	Jean-Marc Baillard
65	DIGT	4k	DigitPac ROM	Engineering	Ángel Martin
66	DIIL	4k	HP-IL Diagnostics	Extensions	HP Co.
67	DMND	4k	Diamond ROM	Financial	??
68	DYRK	4k	Dyerka ROM	Utilities	David Yerka
69	E41S	8k	ES41 Module	Extensions	Eramco
70	EENG	4k	Electrical Eng.	Engineering	HP Co. + ÁM
71	EILP	4k	Extended IL+	Extensions	Christoph Klug
72	EPRH	4k	ML Eprom 1H	Utilities	S. Bariziene & JJ Dhenin
73	EPRM	4k	MM Eprom	Utilities	Dutch PPC Members
74	ESML	4k	ES MLDL 7B	Extensions	Eramco
75	ETI3	4k	ETSII-3	Engineering	Ángel Martin
76	ETI4	4k	ETSII-4	Engineering	Ángel Martin
77	ETI5	4k	ETSII-5	Engineering	Ángel Martin
78	EXIO	4k	Extended I/O Module	Extensions	HP Co.
79	EXTI	4k	Extended-IL ROM	Extensions	Ken Emery
80	FACC	4k	300889_FACC	Engineering	??
81	FAIR	8k	Gear Design Module	Engineering	Fairfield Inc.
82	FDYN	4k	Fluid Dynamics Solutions	Engineering	HP Co.
83	FFEE	4k	For-Fee	Engineering	Ángel Martin
84	FINA	4k	Financial Pac	Financial	HP Co.
85	FRTH (*)	8k	FORTH Module	Extensions	Serge Vaudenay
86	FUNS	8k	Fun Stuff Module	Games	Ángel Martin
87	GAME	4k	Games Pac	Games	HP Co.
88	GEOG	4k	Geometry	Math	HP Co. + ÁM
89	GMAS	4k	Auto Fiance-2 Module	Financial	GMAC
90	GMAT	8k	Auto Fiance-3 Module	Financial	GMAC
91	GMTY	4k	Geometry 2011	Math	Jean-Marc Baillard
92	HCMP	4k	HydraComp ROM	Engineering	Paul Monroe
93	HEPR	4k	HEPAX RAM Template	Extensions	VM Electronics
94	HEPX	16k	HEPAX Module	Extensions	VM Electronics
95	HMTH	4k	High-Level Math	Math	HP Co.
96	HOME	4k	Home Management. Pac	Financial	HP Co.
97	HVAC	4k	Heatinv, Vent. & AirCon	Engineering	HP Co.
98	ICDO	4k	Icode ROM	Extensions	??
99	ICEB	4k	ICE Box	Utilities	Geir Isene
100	IDC1	8k	ML-ICD	Engineering	BCMC 1987
101	IDC2	4k	BG/UG IDC	Engineering	BCMC 1985

#	ID	Size	Name	Type	Author
102	IERF	8k	Inverse ERF	Math	Ángel Martin
103	ILBF	4k	IL-Buffer	Extensions	Ángel Martin
104	IMDB	4k	Module Database	Extensions	Monte Dalrymple
105	INTG	8k	Integrator ROM	Math	JM Baillard & Á. Martin
106	ISEN	4k	ISENE ROM	Utilities	Geir Isene
107	ISOL	4k	Interchangeable. Solutions	Science	UPLE
108	JMAT	8k	JMB Math	Math	Jean-Marc Baillard
109	JMTX	8k	JMB Matrix	Math	Jean-Marc Baillard
110	KC135	12k	Weight & Balance Comp.	Engineering	??
111	L119	8k	AFDC-1E-003	Engineering	Zengun
112	LAIT (*)	4k	LaitRAM XQ2	Extensions	LaitRam Corp.
113	LAND	4k	Land Navigation ROM	Science	Warren Furlow
114	LBLS	4k	Labels ROM	Extensions	W. Doug Wilder
115	LENG	4k	Solar Engineering Solutions	Engineering	HP Co.
116	LNDL	4k	Lend, Lease & Sav.	Financial	HP Co.
117	LPLC	8k	Laplace Transform ROM	Math	Raymond Moore
118	MADV	12k	Advanced Matrix Pac	Math	Ángel Martin
119	MASS	4k	Mass Storage Utilities	Extensions	Ángel Martin
120	MATH	4k	Math Pac	Math	HP Co.
121	MCHN	4k	Machine Construction Pac	Engineering	HP Co.
122	MCMP	4k	Mountain Computer	Utilities	Paul Lind
123	MDP1	8k	AFDC-1F ROM	Engineering	Zengun
124	MDP2	8k	AFDC-1F ROM	Engineering	Zengun
125	MELB	4k	Melbourne ROM	Utilities	PPC Members
126	MENG	4k	Mechanical Eng.	Engineering	HP Co. + ÁM
127	MILE	8k	Military Engineering ROM	Engineering	??
128	MLBL	4k	Mainframe Labels	Extensions	David van Leeuwen
129	MLRM	4k	ML ROM	Utilities	Frits Ferwerda
130	MLTI	8k?	Multi-Prec. Library	Math	Peter Platzer
131	MONO	8k	MONOPOLY ROM	Games	Thomas Rodke
132	MTRX	4k	MATRIX ROM	Math	Ángel Martin
133	MTST	4k	MC Test ROM	Utilities	??
134	MUEC	8k	Muecke ROM	Engineering	Mücke Software GmbH
135	NAVI	8k	Navigation Pac	Science	HP Co.
136	NCHP	4k	NoVoCHAP	Utilities	G. Isene & A. Martin
137	NEA1	8k	SNEAP1	Engineering	SNEAP Society (F)
138	NEA2	8k	SNEAP2	Engineering	SNEAP Society (F)
139	NEA3	8k	SNEAP3	Engineering	SNEAP Society (F)
140	NFCR	4k	NFC ROM	Utilities	Nelson F. Crowle
141	NPAC	8k	NavPac ROM	Science	??
142	NTHY	4k	Number Theory ROM	Math	Jean-Marc Baillard
143	NVCM	8k	NaVCOM 2	Science	??
144	OILW	8k	OilWell Module	Engineering	Jim Daly
145	OPTO	4k	Optometry I & II	Science	HP Co.
146	OTRP	8k	OvenTrop	Engineering	WMK
147	P3BC	16k	Aviation for P3B/C	Engineering	??
148	PANA	8k	PANAME ROM	Utilities	S. Barziene & JJ Dhenin
149	PARI	4k	PARIO ROM	Extensions	Nelson F. Crowle
150	PCOD	4k	Proto-Coder 1A	Extensions	Nelson F. Crowle
151	PETR	8k	Petroleum Pac	Engineering	HP Co.
152	PHYH	4k	Physics	Science	HP Co.
153	PLOT	8k	Plotter Module	Extensions	HP Co.
154	PMLB	4k	PPC Melb ROM	Utilities	PPC Members

#	ID	Size	Name	Type	Author
155	POLY	8k	Polynomial Analysis	Math	Á. Martin & JM Baillard
156	PPCM	8k	PPC ROM	Utilities	PPC Members
157	PRFS	4k	ProfiSet	Extensions	Winfried Maschke
158	PRIQ	8k	PRIDE ROM	Engineering	James Friesing
159	PWRL	16k	PowerCL_B4	Extensions	Ángel Martin
160	QUAT	8k	Quaternion ROM	Math	Jean-Marc Baillard
161	RAMP	4k	RAMPPage Module	Utilities	Ángel Martin
162	REAL	8k	Real State Pac	Financial	HP Co.
163	RM32	4k	RAMBOX-32	Extensions	W&W GmbH
164	ROAM	4k	ROAM Module	Utilities	Wilson B. Holes
165	ROMS	4k	SV's ROM	Utilities	Serge Vaudenay
166	SANA	12k	SandMath-12k	Math	Ángel Martin
167	SBOX	8k	SandBox	Utilities	Ángel Martin
168	SEAK	4k	SeaKing MK5	Engineering	Navy Air
169	SECY	4k	Securities Pac	Financial	HP Co.
170	SGSG	4k	Gas Module	Engineering	SGS Redwood
171	SIHP	4k	Solve/Intg ROM	Math	HP Co. + ÁM
172	SIMM	16k	SIM Module	Science	??
173	SKWD	4k	Skwid's BarCode	Extensions	Ken Emery
174	SMCH	8k	Speed Machine	Financial	Alameda Mngmt. Corp.
175	SMPL	4k	Simplex Module	Math	Phillipe J. Roussel
176	SMTS	8k	SandMath-8k	Math	Ángel Martin
177	SND2	8k	SandMath-II	Math	Ángel Martin
178	SPEC	4k	Spectral Analysis	Math	Jean-Marc Baillard
179	SRVC (*)	4k	Service ROM	Extensions	HP Co.
180	STAN	4k	Standard Pac	Science	HP Co.
181	STAT	4k	Statistics Pac	Math	HP Co.
182	STRE	4k	Stress Analysis Pac	Engineering	HP Co.
183	STRU	8k	Structural An, Pac	Engineering	HP Co.
184	SUD1	4k	SUDOKU & Sound	Games	Á. Martin & JM Baillard
185	SUPR	8k	SUP-R-ROM	Science	James W. Vick
186	SURV	4k	Surveying Pac	Science	HP Co.
187	TEST	4k	Test Statistics	Math	HP Co.
188	THER	4k	Thermal Pac	Engineering	HP Co.
189	TIME	4k	Timer SolBook	Extensions	HP Co.
190	TOMS	4k	Tom's ROM	Science	Thomas A. Bruns
191	TOOL	4k	ToolBox-II	Utilities	Ángel Martin
192	TREK	4k	Start Trek	Games	Ángel Martin
193	TRIH	4k	83Trinh	Utilities	Phil Trinh
194	UNIT	4k	Unit Conversion	Extensions	Ángel Martin
195	USPS	8k	Mail Delivery	Engineering	USPS
196	VECT	4k	VECTOR Analysis	Math	Ángel Martin
197	WMK3	8k	WMK-3	Engineering	WMK
198	WMK4	4k	WMK-4	Engineering	WMK
199	XXXA	4k	Empty		Not listed
200	XXXB	4k	Empty		Not listed
201	XXXC	4k	Empty		Not listed
202	XXXD	8k	Empty		Not listed
203	XXXE	8k	Empty		Not listed
204	XXXF	16k	Empty		Not listed
205	YACH	8k	Yach Computer	Science	Bobby Schenk
206	YFNP	4k	YFNS Plus	Extensions	Monte Dalrymple
207	YFNS	4k	Alternate YFNS	Extensions	Monte Dalrymple

#	ID	Size	Name	Type	Author
208	YFNZ	4k	Main YFNS	Extensions	Monte Dalrymple
209	Z41Z	8k	41Z Module	Math	Ángel Martin
210	ZENR	4k	Zenrom	Utilities	Zengrange Ltd.
211	ZEPR	4k	Programmer	Extensions	Zengrange Ltd.

(*)Take-over ROM. Need to use PPG#4

(**)Must be located in page#4. Plug it manually using YPOKE

Other modules not included in the Library:-

For sure many more of these abound, yet these are the ones I know about – feel free to complete the list with your own entries, and don't forget to share them with the whole community.

1. Market Forecast 4k Forecaster?
2. Mortar Fire Data Calculator 8k MDN Canada
3. Dr. Z RaceTrack Module 4k William T. Ziembra



© Photo courtesy of Wilson "Bill" Holes.

Appendix 2. FOCAL program Listings.

Provided for your reference and in case you feel like experimenting with your own settings.

As always, mind the potential conflicts with other modules when plugging stuff, and pay special attention not to overwrite YFNS. (you're safe if using **PLUGGX** – it won't let you to :-)

In the HEPAX configuration code the role of **HEPINI** is to write the appropriate words into the HRAM pages, as per the description provided before. This could also be done using **YPOKE**, but the memory requirements are much larger due to all the alpha strings that would be required to do so.

For example, see below for the 16k case, using pages C,D,E, and F.

This would mean having to write on each page the four page id#s, plus the pointers to the previous and next pages, for a total of 10x – or equivalent to 110 bytes:

"809FE7-000C"
"808000-000C"

"808FE8-000D"
"80AFE7-000D"
"809000-000D"

"809FE8-000E"
"80BFE7-000E"
"80A000-000E"

"80AFE8-000F"
"80B000-000F"

In fact such was the original method used in earlier versions of the CLUTILS – so using **HEPINI** resulted in significant byte savings that allowed the inclusion of other functionality.

01	LBL "HPX4"	Entry for 4k
02	E	
03	GTO 02	
04	LBL "HPX8"	Entry for 8k
05	2	
06	GTO 02	
07	LBL "HPX16"	Entry for 16k
08	0	
09	LBL 02	
10	X<>F	flag setting
11	TURBO50	run as fas as possible
12	"0B9>808"	prepare for 0x808
13	AVIEW	provide feedback
14	YMCPY	copy to 0x808 in RAM
15	FS? 00	4k case?
16	GTO 00	yes, skip the rest
17	YBSP	remove last chr
18	"9"	replace it
19	AVIEW	provide feedback
20	YMCPY	copy to 0x809 in RAM
21	FS? 01	8k case?
22	GTO 00	yes, skip the rest
23	YBSP	remove last char
24	"A"	replace it
25	AVIEW	provide feedback
26	YMCPY	copy to 0x80A in RAM
27	YBSP	remove last char
28	"B"	replace it
29	AVIEW	provide feedback
30	YMCPY	copy to 0x80B in RAM
31	LBL 00	
32	"SETUP..."	announce last phase
33	AVIEW	
34	"-RAM"	buffer common string
35	ASTO L	in temporary storage
36	"808"	build first mapping text
37	ARCL L	
38	PLUG4U	plug 0x808 to page#F
39	FS? 00	4k case?
40	GTO 01	yes, jump over
41	"809"	build 2nd. Mapping text
42	ARCL L	
43	PLUG4L	plug 0x809 to page#E
44	FS? 01	8k case?
45	GTO 03	yes, jumpm over
46	"80A"	build 3er. Mapping text
47	ARCL L	
48	PLUG3U	plug 0x80A to page#D
49	"80B"	build 4th. Mapping text
50	ARCL L	
51	PLUG3L	plug 0x80B to port#C
52	4	configuration parameters:
53	ENTER^	four pages
54	12	starting at page#C
55	GTO 00	
56	LBL 03	8k case
57	2	config parm:
58	ENTER^	two pages
59	14	starting at page#E
60	GTO 00	
61	LBL 01	4k case
62	E	config parm:
63	ENTER^	just one page,
64	15	starting at page#F
65	LBL 00	
66	HEPINI	configure HEPAX FileSys
67	E	get page# below
68	-	
69	"HEPX"	
70	PLUGGX	plug HEPAX there
71	HEPDIR	show we've done it
72	END	done.

Appendix 3.- MCODE Listing showing the Alphabetical sections prompting code.

The function **CLLIB** begins by building the prompt text in the display. Using the OS routine [PROMF2] is helpful to save bytes, so there's no need to write the function name again. Alpha is cleared using [CLA], just to prepare for a possible copy of the ROM id# to Alpha using the [A] hot-key in run mode. Then we get into a partial data entry "condition", waiting for a key to be pressed.

Back Arrow sends the execution to [EXIT3], to do the housekeeping required to reset everything back to the standard OS-required status (disable Display, resetting Keyboard bits, CPU flags, etc.). Since the valid keys are quite a lot [A-Z] we need to use multiple conditions in the logic. The first two rows are the easiest; as they set up CPU flag#4 and that can be tested easily. In this case we copy the mantissa sign in A to C[S&X], then store it in B[S&X] and we move on.

1	CLLIB	BCKARW	A66E	341	PORT DEP:	
2	CLLIB		A66F	08C	GO	
3	CLLIB		A670	36E	->A36E	[EXIT3]
4	CLLIB	Header	A671	082	"B"	
5	CLLIB	Header	A672	009	"J"	
6	CLLIB	Header	A673	00C	"L"	
7	CLLIB	Header	A674	00C	"L"	
8	CLLIB	Header	A675	003	"C"	CL Library Prompting - Alphabetical
9	CLLIB	CLLIB	A676	000	NOP	
10	CLLIB		A677	158	M=C ALL	
11	CLLIB		A678	345	?NC XQ	Clears Alpha [CLA]
12	CLLIB		A679	040	->10D1	
13	CLLIB		A67A	3C1	?NC XQ	Enable & Clear Display [CLLCDE]
14	CLLIB		A67B	0B0	->2CF0	
15	CLLIB		A67C	198	C=M ALL	fst id#
16	CLLIB		A67D	34D	?NC XQ	
17	CLLIB		A67E	014	->05D3	[PROMF2]
18	CLLIB		A67F	3BD	?NC XQ	
19	CLLIB		A680	01C	->07EF	[MESSL]
20	CLLIB		A681	001	"A"	
21	CLLIB		A682	02D	"."	
22	CLLIB		A683	21A	"Z"	
23	CLLIB		A684	115	?NC XQ	Partial Data Entry! [NEXT1]
24	CLLIB		A685	038	->0E45	
25	CLLIB		A686	343	JNC -24d	
26	CLLIB		A687	04C	?FSET 4	rows 1 & ??
27	CLLIB		A688	063	JNC +12d	
28	CLLIB		A689	130	LDI S&X	
29	CLLIB		A68A	00A	CON:	
30	CLLIB		A68B	35E	?A#0 MS	was "J" pressed?
31	CLLIB		A68C	023	JNC +04	yes, skip next
32	CLLIB		A68D	046	C=0 S&X	
33	CLLIB		A68E	0BE	A<->C MS	
34	CLLIB		A68F	2FC	RCR 13	
35	CLLIB		A690	0E6	C<->B S&X	save chr# in B[S&X]
36	CLLIB		A691	369	PORT DEP:	Transfer to next section
37	CLLIB		A692	03C	GO	
38	CLLIB		A693	2E9	->A6E9	[MOVEON!]
39	CLLIB		A694	0B0	C=N ALL	pressed keycode
40	CLLIB		A695	106	A=C S&X	save in A[S&X] for comparisons
41	CLLIB		A696	21C	PT= 2	adjust pointer
42	CLLIB		A697	05A	C=0 M	reset counter
43	CLLIB		A698	130	LDI S&X	upper value
44	CLLIB		A699	01A	"Z"	from there down!
45	CLLIB		A69A	1BC	RCR 11	put it in C[M]
46	CLLIB		A69B	013	JNC +02	skip over line
47	CLLIB		A69C	343	JNC -24d	pipe-lining up

For the rest [K-Z] we'll need to read the keycode of the pressed key and act accordingly. Also *we need to discard any non-letter key*, rejecting it if its keycode value is outside of the [A,Z] range.

Now the show is about to start: see how the key pressed value (in N) is compared with every possible value in the [K-Z] range, building the "pointer" in C[S&X] by repeat one-additions until coming up to its final result.

48	CLLIB		A69D	130	LDI S&X	
49	CLLIB		A69E	120	CON:	XEQ keycode [120]
50	CLLIB		A69F	366	?A#C S&X	
51	CLLIB		A6A0	1C3	JNC +56d	[K]
52	CLLIB		A6A1	222	C=C+1 @PT	keycode [220]
53	CLLIB		A6A2	366	?A#C S&X	
54	CLLIB		A6A3	1B3	JNC +54d	[L]
55	CLLIB		A6A4	222	C=C+1 @PT	keycode [320]
56	CLLIB		A6A5	366	?A#C S&X	
57	CLLIB		A6A6	1A3	JNC +52d	[M]
58	CLLIB		A6A7	130	LDI S&X	
59	CLLIB		A6A8	030	CON:	ENTER^ keycode [030]
60	CLLIB		A6A9	366	?A#C S&X	
61	CLLIB		A6AA	18B	JNC +49d	[N]
62	CLLIB		A6AB	222	C=C+1 @PT	keycode [130]
63	CLLIB		A6AC	222	C=C+1 @PT	keycode [230]
64	CLLIB		A6AD	366	?A#C S&X	
65	CLLIB		A6AE	173	JNC +46d	[O]
66	CLLIB		A6AF	222	C=C+1 @PT	keycode [330]
67	CLLIB		A6B0	366	?A#C S&X	
68	CLLIB		A6B1	163	JNC +44d	[P]
69	CLLIB		A6B2	130	LDI S&X	
70	CLLIB		A6B3	040	CON:	"-" keycode [040]
71	CLLIB		A6B4	366	?A#C S&X	
72	CLLIB		A6B5	14B	JNC +41d	[Q]
73	CLLIB		A6B6	222	C=C+1 @PT	keycode [140]
74	CLLIB		A6B7	366	?A#C S&X	
75	CLLIB		A6B8	13B	JNC +39d	[R]
76	CLLIB		A6B9	222	C=C+1 @PT	keycode [240]
77	CLLIB		A6BA	366	?A#C S&X	
78	CLLIB		A6BB	12B	JNC +37d	[S]
79	CLLIB		A6BC	222	C=C+1 @PT	keycode [340]
80	CLLIB		A6BD	366	?A#C S&X	
81	CLLIB		A6BE	11B	JNC +35d	[T]
82	CLLIB		A6BF	130	LDI S&X	
83	CLLIB		A6C0	050	CON:	"+" keycode [050]
84	CLLIB		A6C1	366	?A#C S&X	
85	CLLIB		A6C2	103	JNC +32d	[U]
86	CLLIB		A6C3	222	C=C+1 @PT	keycode [150]
87	CLLIB		A6C4	366	?A#C S&X	
88	CLLIB		A6C5	0F3	JNC +30d	[V]
89	CLLIB		A6C6	222	C=C+1 @PT	keycode [250]
90	CLLIB		A6C7	366	?A#C S&X	
91	CLLIB		A6C8	0E3	JNC +28d	[W]
92	CLLIB		A6C9	222	C=C+1 @PT	keycode [350]
93	CLLIB		A6CA	366	?A#C S&X	
94	CLLIB		A6CB	0D3	JNC +26d	[X]
95	CLLIB		A6CC	130	LDI S&X	
96	CLLIB		A6CD	060	CON:	"*" keycode [060]
97	CLLIB		A6CE	366	?A#C S&X	
98	CLLIB		A6CF	0BB	JNC +23d	[Y]
99	CLLIB		A6D0	222	C=C+1 @PT	keycode [160]
100	CLLIB		A6D1	366	?A#C S&X	
101	CLLIB		A6D2	0AB	JNC +21	[Z]
102	CLLIB		A6D3	265	?NC XQ	Blink Display - pass #1
103	CLLIB		A6D4	020	->0899	[BLINK1]
104	CLLIB		A6D5	265	?NC XQ	Blink Display - pass #2
105	CLLIB		A6D6	020	->0899	[BLINK1]
106	CLLIB		A6D7	22B	JNC -59d	ONE PROMPT
107	CLLIB	[K]	A6D8	27A	C=C-1 M	
108	CLLIB	[L]	A6D9	27A	C=C-1 M	
109	CLLIB	[M]	A6DA	27A	C=C-1 M	
110	CLLIB	[N]	A6DB	27A	C=C-1 M	
111	CLLIB	[O]	A6DC	27A	C=C-1 M	
112	CLLIB	[P]	A6DD	27A	C=C-1 M	
113	CLLIB	[Q]	A6DE	27A	C=C-1 M	
114	CLLIB	[R]	A6DF	27A	C=C-1 M	
115	CLLIB	[S]	A6E0	27A	C=C-1 M	
116	CLLIB	[T]	A6E1	27A	C=C-1 M	
117	CLLIB	[U]	A6E2	27A	C=C-1 M	
118	CLLIB	[V]	A6E3	27A	C=C-1 M	
119	CLLIB	[W]	A6E4	27A	C=C-1 M	
120	CLLIB	[X]	A6E5	27A	C=C-1 M	
121	CLLIB	[Y]	A6E6	27A	C=C-1 M	
122	CLLIB	[Z]	A6E7	03C	RCR 3	place it in C[S&X]
123	CLLIB		A6E8	0E6	C<>B S&X	save chr# in B[S&X]

The last part is about presenting the chosen key – allowing NULLing if it's held down long enough – Resetting everything back to normal conditions [CLNUP], and see whether there actually exists such a section – before we launch into a blindfold enumeration. This is done by the subroutine [SRCHR], which will fetch the address in the ROM id #table where the section starts. With that we'll transfer the execution to the **ROMLIB** function code where the actual enumeration will take place - only with a padded value to start from, as opposed to doing it from the top of the table.

124	CLLIB	MOVEON	A6E9	379	PORT DEP:	CleanUp and Show
125	CLLIB		A6EA	03C	XQ	Allows NULLing
126	CLLIB		A6EB	261	->A661	[CLNUP]
127	CLLIB		A6EC	149	?NC XQ	Disable PER, enable RAM
128	CLLIB		A6ED	024	->0952	[ENCP00]
129	CLLIB		A6EE	215	?NC XQ	Reset BIT sequence
130	CLLIB		A6EF	00C	->0385	[RSTSQ]
131	CLLIB		A6F0	130	LDI S&X	Location of first ID
132	CLLIB		A6F1	32B	<-start of search>	[romtbl]
133	CLLIB		A6F2	106	A=C S&X	save offset in A[S&X]
134	CLLIB		A6F3	349	PORT DEP:	Search Char in B[S&X]
135	CLLIB		A6F4	08C	XQ	start offset in A[S&X]
136	CLLIB		A6F5	353	->A353	[SRCHR]
145	CLLIB		A6F6	023	JNC +04	not found, bail out with style
146	CLLIB		A6F7	341	PORT DEP:	Transfer code to Main LIB
147	CLLIB		A6F8	08C	GO	
148	CLLIB		A6F9	386	->A386	[MERGE]
149	CLLIB	INFMSG	A6FA	261	?NC XQ	debounce keyboard
150	CLLIB		A6FB	000	->0098	[RSTKB]
151	CLLIB		A6FC	3B5	PORT DEP	Display Message
152	CLLIB		A6FD	08C	XQ	Unless Error Flag is set.
153	CLLIB		A6FE	3EE	->AFEE	[DSPERR]
154	CLLIB		A6FF	00E	"N"	
155	CLLIB		A700	00F	"O"	
156	CLLIB		A701	020	" "	
157	CLLIB		A702	013	"S"	
158	CLLIB		A703	015	"U"	
159	CLLIB		A704	003	"C"	
160	CLLIB		A705	208	"H"	
161	CLLIB		A706	389	PORT DEP	Output Message
162	CLLIB		A707	08C	GO	
163	CLLIB		A708	016	->A816	[APEREX]

Note how [SRCHR] is really part of the **ADRID** function code, which also does table look-ups for its own purpose. This code is written around the table structure; refer to the Blueprints for more details.

32	ADRID	SRCHR	A353	04E	C=0 ALL	Expects chr# in B[S&X]
33	ADRID		A354	35D	?NC XQ	
34	ADRID		A355	000	->00D7	[PCTOC]
35	ADRID		A356	03C	RCR 3	get page number
36	ADRID		A357	130	LDI S&X	
37	ADRID		A358	300	CON:	
38	ADRID		A359	1E6	C=C+C S&X	double it up
39	ADRID		A35A	206	C=C+A S&X	add offset to it
40	ADRID		A35B	1BC	RCR 11	put it in C(6:3)
41	ADRID		A35C	0BA	A<>C M	put it in A(6:3)
42	ADRID		A35D	066	A<>B S&X	put reference in A[S&X]
43	ADRID	NXTADR	A35E	0BA	A<>C M	bring adr to C[M]
44	ADRID		A35F	11A	A=C M	keep it in A[M]
45	ADRID		A360	330	FETCH S&X	read value at address
46	ADRID		A361	2E6	?C#0 S&X	value non-zero?
47	ADRID		A362	3A0	?NC RTN	NO, TERMINATE HERE!
48	ADRID		A363	366	?A#C S&X	are they different?
49	ADRID		A364	033	JNC +06	no, -> [FOUND]
50	ADRID		A365	05A	C=0 M	add offset: 5 BYTES
51	ADRID		A366	01C	PT=3	
52	ADRID		A367	150	LD@PT- 5	
53	ADRID		A368	15A	A=A+C M	next addr field
54	ADRID		A369	3AB	JNC -11d	loop back
55	ADRID	FOUND	A36A	1B0	POPADR	
56	ADRID		A36B	23A	C=C+1 M	
57	ADRID		A36C	170	PUSHADR	increase RTN adr
58	ADRID		A36D	3E0	RTN	

And that's all folks - easy when you know the tricks ☺

Appendix 4.- Serial Transfer CLWRITE and CLREAD source code.

- Written by Raymond Wiker.

1) Copy YFNS-1A to RAM at 80C000, and patch for items 2, 5, 8. These affect the operation of YIMP. I did this with a variation of the PATCHIT program posted earlier.

2) Execute TURBO50.

3) Execute SERINI

4) Execute BAUD12. From the documentation, this should not be necessary, but I had to explicitly set 1200 baud to get the transfer to work.

5) The file yfns-1e.rom has the opposite byte order of what YIMP expects, so the transfer program needs to perform byte swapping. Alternatively, you might do the byte-swapping before you do the transfer.

6) Transfer the ROM; I chose to transfer it to 80D000 (i.e, put 80D000-0FFF in the alpha register, start YIMP). For the transfer, I used the CLWriter program that I posted a few days back, with the command

```
CLWriter.exe yfns-1e-fixed.rom com1 1200 5
```

```
--- the file yfns-1e-fixed.rom is the byte-swapped version of  
yfns-1e.rom. I probably should have chosen a slightly different  
name, but that does not really matter. The "5" means that I put a 5  
millisecond delay after each byte. It may not actually be necessary;  
I added it because I got timeouts, but these were probably because I  
left out step 4 (BAUD12).
```

7) Execute **PLUG1L** with "80D-RAM" in the Alpha register.

8) Verify (using CATALOG 2) that I'm now running YFNS-1E.

CLREADER follows somehow the reverse process, Make sure you execute **CLREADER** on the PC first, followed by **YEXP** so there won't be any data stream missed.

Using TURBO50 settings the transfer rate can safely be increased to 4800 and most likely up to 9600 without any data loss. More critical is the padding time, which heavily depends on your system hardware. I have obtained reliable results with 1 ms and 4800 as my standard settings in either direction - write and read.

```
using System;
using System.IO;
using System.IO.Ports;
using System.Threading;

public class CLWriter
{
    public static void Main(string [] args)
    {
        int baudrate = 1200;
        int delay = 0;
        if (args.Length < 2) {
            Console.Error.WriteLine("Usage:");
            Console.Error.WriteLine("  {0} file port [baudrate [delay]]", "CLWriter");
            Console.Error.WriteLine();
            Console.Error.WriteLine("Where baud defaults to {0}", baudrate);
            Console.Error.WriteLine("and delay defaults to {0}", delay);
            Console.Error.WriteLine("Available Ports:");
            Console.Error.WriteLine();
            foreach (string s in SerialPort.GetPortNames())
            {
                Console.Error.WriteLine("  {0}", s);
            }
            return;
        }
        string filename = args[0];
        string portname = args[1];
        if (args.Length > 2) {
            baudrate = int.Parse(args[2]);
            if (baudrate != 1200 && baudrate != 2400 &&
                baudrate != 4800 && baudrate != 9600) {
                Console.Error.WriteLine("Invalid baudrate {0}; should be one of", baudrate);
                Console.Error.WriteLine("1200, 2400, 4800, 9600");
                return;
            }
        }
        if (args.Length > 3) {
            delay = int.Parse(args[3]);
            if (delay > 10) {
                Console.Error.WriteLine("delay {0} probably too large.", delay);
                return;
            }
        }

        if (!File.Exists(filename)) {
            Console.Error.WriteLine("File {0} does not exist.", filename);
            return;
        }

        FileStream fstream = File.Open(filename, FileMode.Open);

        if (fstream.Length > 8192) {
            Console.Error.WriteLine("WARNING: {0} is over 8192 bytes long ({1});", filename,
fstream.Length);
            Console.Error.WriteLine("Will only transfer the first 8192 bytes.");
        }

        BinaryReader binReader = new BinaryReader(fstream);
```

```
SerialPort serialport = new SerialPort();
serialport.PortName = portname;
serialport.BaudRate = baudrate;
serialport.Parity = Parity.None;
serialport.DataBits = 8;
serialport.StopBits = StopBits.One;
serialport.Handshake = Handshake.None;

serialport.Open();

try {
    byte[] buffer = new byte[8192];
    int count = binReader.Read(buffer, 0, 8192);

    // swap high & low bytes:
    for (int i = 0; i < count; i+= 2) {
        byte tmp = buffer[i];
        buffer[i] = buffer[i+1];
        buffer[i+1] = tmp;
    }

    for (int i = 0; i < count; i++) {
        Console.Write("{0:x2} ", buffer[i]);
        if (i % 16 == 15) {
            Console.WriteLine();
        }
        serialport.Write(buffer, i, 1);
        if (delay > 0) {
            Thread.Sleep(delay);
        }
    }
    Console.WriteLine();
}
catch (EndOfStreamException) {
    // nada
}
serialport.Close();
}
```

```
using System;
using System.IO;
using System.IO.Ports;
using System.Threading;

public class CLReader
{
    public static void Main(string [] args)
    {
        int baudrate = 1200;
        if (args.Length < 2) {
            Console.Error.WriteLine("Usage:");
            Console.Error.WriteLine("  {0} file port [baudrate]", "CLReader");
            Console.Error.WriteLine();
            Console.Error.WriteLine("Where baud defaults to {0}", baudrate);
            Console.Error.WriteLine("Available Ports:");
            Console.Error.WriteLine();
            foreach (string s in SerialPort.GetPortNames())
            {
                Console.Error.WriteLine("  {0}", s);
            }
            return;
        }
        string filename = args[0];
        string portname = args[1];
        if (args.Length > 2) {
            baudrate = int.Parse(args[2]);
            if (baudrate != 1200 && baudrate != 2400 &&
                baudrate != 4800 && baudrate != 9600) {
                Console.Error.WriteLine("Invalid baudrate {0}; should be one of", baudrate);
                Console.Error.WriteLine("1200, 2400, 4800, 9600");
                return;
            }
        }
        if (File.Exists(filename)) {
            Console.Error.WriteLine("File {0} already exists, will not overwrite.",
                filename);

            return;
        }
        SerialPort serialport = new SerialPort();
        serialport.PortName = portname;
        serialport.BaudRate = baudrate;
        serialport.Parity = Parity.None;
        serialport.DataBits = 8;
        serialport.StopBits = StopBits.One;
        serialport.Handshake = Handshake.None;
        serialport.ReadTimeout = 30000; // milliseconds

        serialport.Open();

        byte[] buffer = new byte[8192];
        int pos = 0;

        try {
            int count;

            do {
                count = serialport.Read(buffer, pos, 8192 - pos);
                pos += count;
            }
        }
    }
}
```



```
        } while (pos < 8192);
    }
    catch (TimeoutException) {          // nada
    }
    serialport.Close();

    if (pos % 2 != 0) {
        Console.Error.WriteLine("Odd number of bytes read.");
        return;
    }
    // swap high & low bytes:
    for (int i = 0; i < pos; i+= 2) {
        byte tmp = buffer[i];
        buffer[i] = buffer[i+1];
        buffer[i+1] = tmp;
    }

    FileStream fstream = File.Open(filename, FileMode.CreateNew, FileAccess.Write);

    BinaryWriter binWriter = new BinaryWriter(fstream);

    binWriter.Write(buffer, 0, pos);
        binWriter.Close();
    fstream.Close();

}
}
```