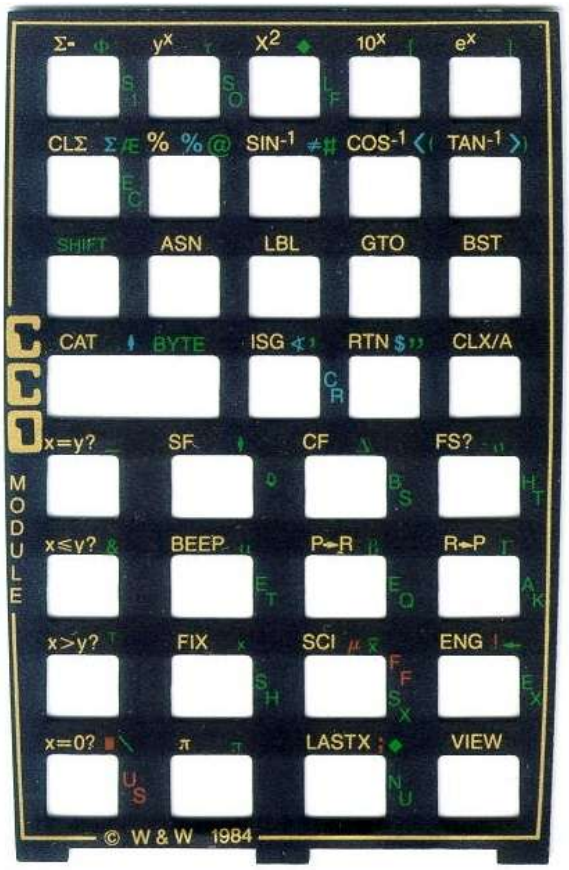
AMC_OS/X Module

System Extensions for the HP-41CX
Revision – 4M

User's Manual and QRG.



Written and Programmed by Ángel M. Martin

July 2014

This compilation revision 4.W.6.6

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

Acknowledgments.- This manual and the AMC_OS/X module would obviously not exist without the CCD Module. Thanks to Raymond del Tondo for the initial OS/X 4k-footprint version with much of the OS extensions extracted from the CCD. Also thanks to Håkan Thörngren, Fritz Ferwerda and Nelson F. Crowle for their powerful functions, examples of solid MCODE programming incorporated to this module.

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.

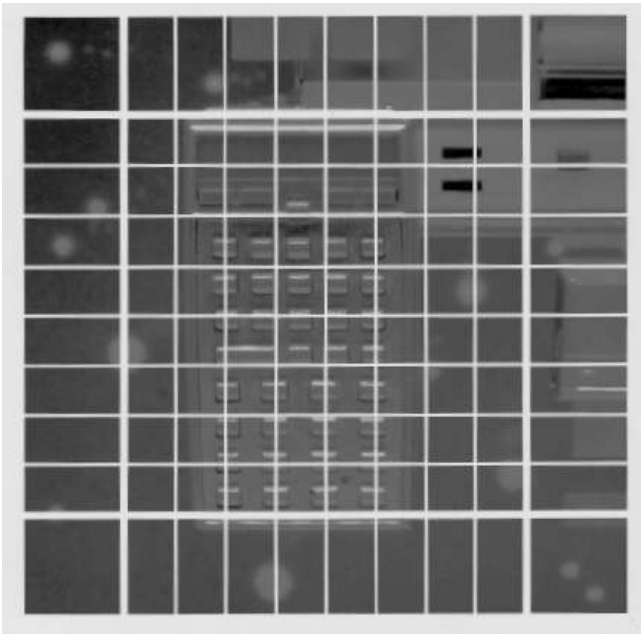
AMC_OS/X Module

Table of Contents.

7. System Advanced Utilities

8. AECROM Program Generator

Appendices.



AMC_OS/X Module
OS Extension for the HP-41CX

1. Introduction.

Of the several modules that included extensions for the operation system, the CCD Module was no doubt the most daring in the implementation and useful in the results. The original module featured numerous enhancements that made using the powerful features of the 41 a much easier affair; plus added a few more on its own to round out the functionality. Only the ZENROM came close to a similar set of capabilities, although to this author not quite as well thought out and lacking some finesse in the integration.

Even as groundbreaking as the CCD Module was, yet further usability came from a reduced footprint version that removed other functions not directly related to the OS extensions. Raymond del Tondo produced the first of these 4k modules, the CCD_OS/X that added a couple of functions to read/write module images to the HP-IL mass storage.

That was the perfect basis to extend using the Library#4 – moving sections of the code to the library space, making more room for additional features and a few more FAT entries to allocate additional functions. The first AMC_OS/X was just that, a 4k-module adding on the CCD_OS/X the following:

- Complete implementation of the CCD Catalogs
- Added 5 new XM File types (to be described later on)
- Prompt Lengthener from the ML ROM
- Multi-byte assign (ASG) from the ML ROM
- Direct GTO to ROM Address
- Plus 15+ more utility functions

Page#4 Library and Bank-Switching.

A subsequent version of the module added bank-switching support, which allowed for a substantial increase in the number of functions. Several utilities from the ToolBox and Rampage ROMs were added to the set, so at this stage the module was fulfilling its goal of being a permanent fixture in all systems, with only minimal footprint requirements. It was also modified to be compatible with pages #6 and #7, assuming of course that no printer or HL-IL module are plugged in.

Amongst the added functions you'll find the usual suspects: Buffer and Page Catalogs, Buffer and KA Save/Write to X-Mem; Focal program compiler, X-Mem write-to / read-from HP-IL disk file, Checksum Page summing and other X-Mem file utilities.

The last touch was the addition of a third bank, including the AECROM Program Generator – arguably the first CAS-like approach even if in proto-embryonic shape. You may not find it very useful nowadays but it remains a world-class example of MCODE programming. Consider that it takes more than 3k of ROM space, the third bank is pretty full. Porting the original code from the AECROM to a bank-switched implementation was a challenge but also a very rewarding project - and certainly a lot of fun.

Remember: The AMC_OS/X extensively uses routines and functions from the Page#4 Library. Make sure the Library#4 revision "K" (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.

Function index at a glance.

Without further ado, here are all AMC_OS/X functions: a full-house FAT with the best tools in town.

Function	Description	Input	Output	Author
-AMC"OS/X	*Section Headar*	*None*	*Splash Screen*	*Nelson F. Crowle*
ABSP	Alpha Back Space	Text in Alpha	Deletes Rightmost chr	*W&W GmbH*
ARCLH	ARCL Hex	number in X-reg	added to Alpha in HEX	*W&W GmbH*
ARCLI	ARCL Integer	number in X-reg	integer part to ALPHA	*W&W GmbH*
ASG _	Multi-byte Assign	Prompts for data	Asisgns function	*Fritz Ferwerda*
ASWAP	Alpha Swap	A,B in Alpha	B,A in Alpha	*Ángel Martin*
B?	Buffer presence test	buffer id# in X-reg	YES/NO, skips if false	*W&W GmbH*
CDE	Code	string in ALPHA	NNN in X-reg	*Ken Emery*
CLA-	CLA from blank	String in ALPHA	Deletes chars after blank	*W&W GmbH*
CLB	Clear Buffer	buffer id# in X-reg	Deletes buffer from I/O	*W&W GmbH*
CLEM	Clear X-Mem	None	Deletes all X-Memory	*Håkan Thörngren*
DCD	Decode	NNN in X-reg	String in ALPHA	*Ken Emery*
DTOA	Display to ALPHA	text in display	Text copied to ALPHA	*Ángel Martin*
DTST	Display Test	none	Shows all LCD chars	*Chris Dennis*
F/E	Fix/Eng mode	none	sets both flags	*W&W GmbH*
GTADR _ _ _ _	Go to ROM Address	address as NNN	execution transferred	*W&W GmbH*
LKAX	Suspends/Restores KA	zero/non-zero in X-reg	suspends/activates Local KA	*Ross Cooling*
MNF _ _ _	Mainframe	prompts for fcn id#	executes function	*Clifford Stern*
MSGE _ _ _	Shows OS Message	prompts for msg id#	shows OS message	*Raymond del Tondo*
PC<>RTN	Exchanges PC and RTN	values in PC and RTN stack	values exchanged	*W&W GmbH*
PLNG _	Program Length	Prompts for Name	length in bytes	*W&W GmbH*
PMTA _	prompts in ALPHA	prompts for text	adds text to ALPHA	*W&W GmbH*
PMTH _ _	Prompts for HEX #	prompts for digits	Decimal number in X-reg	*W&W GmbH*
PMTK _	Key Prompt	Prompts for key	branches execution	*W&W GmbH*
RNDM	Random Number	SEED in buffer	random number in X	*W&W GmbH*
SEED	Generates SEED	number in X-reg	creates SEED in buffer	*W&W GmbH*
TAS	Toggles Auto-start	none	Toglles autostart flag - c(16)	*W&W GmbH*
TF _ _	Toggle flag	prompts for flag#	toggles flag status	*Ken Emery*
TGLC	Toggle Lower Case	none	Toggles ALPHA LC mode	*Ángel Martin*
VIEWH	View HEX	number in X-reg	shows HEX in display	*W&W GmbH*
VRG _ _	View Register	prompts for RG#	decodes its contents	*Fritz Ferwerda*
WSIZE _ _	Sets Word size	prompts for size	changes setting	*W&W GmbH*
WSIZE?	Gets Word Size	none	recalls WS to X	*Sebastian Toelg*
XTOAH	X-reg to ALPHA	chr# in X-rg	adds HEX to Alpha	*W&W GmbH*
Y/N? _	Yex/No?	prompts Y/N	branches execution	*PANAME ROM*
PROG _	Program Generator	Prompts formula	Writes FOCAL program	*Nelson F. Crowle*
-OSX BANK2	*ROM List*	*None*	*shows XROM's plugged in*	*Ángel Martin*
BFCAT	Buffer Catalog	none	Enumerates Buffers	*Ángel Martin*
CHKROM _ _	Checks ROM	prompt for ROM id#	Tests OK/BAD chksum	*HP Co.*
CHKSYS	Checks System	none	Tests XROM conflicts	*Ángel Martin*
CMP _	1/2's complement	0,1,2 in prompt	sets complement	*W&W GmbH*
COMPILE	Compiles FOCAL prg	Prog. Name in ALPHA	Compiles GTO/XEQ	*Fritz Ferwerda*
GETBF	Get Buffer	buf. id# in X, FileName in ALPHA	Buffer loaded from X-Mem	*Håkan Thörngren*
GETKA	Get Key Assignments	File Name in ALPHA	KA loaded from X-Mem	*Håkan Thörngren*
HEXIN _	HEX input	prompts for digits	NNN in X-reg	*Håkan Thörngren*
PGSIG _ _	Page Signature	PG# in Prompt	Traininlg Text	*Ángel Martin*
MRGKA	Merge Key Assignments	File Name in ALPHA	Merges keys w/ existing	*Håkan Thörngren*
PEEKR	Peeks Register	absolute adr in X	reg contents to X	*W&W GmbH*
PG? _ _	Page Info	prompts for page#	XROM, #fcns in X	*W&W GmbH*
PGCAT	Page Catalog	none	Enumerates page contents	*VM Electronics*

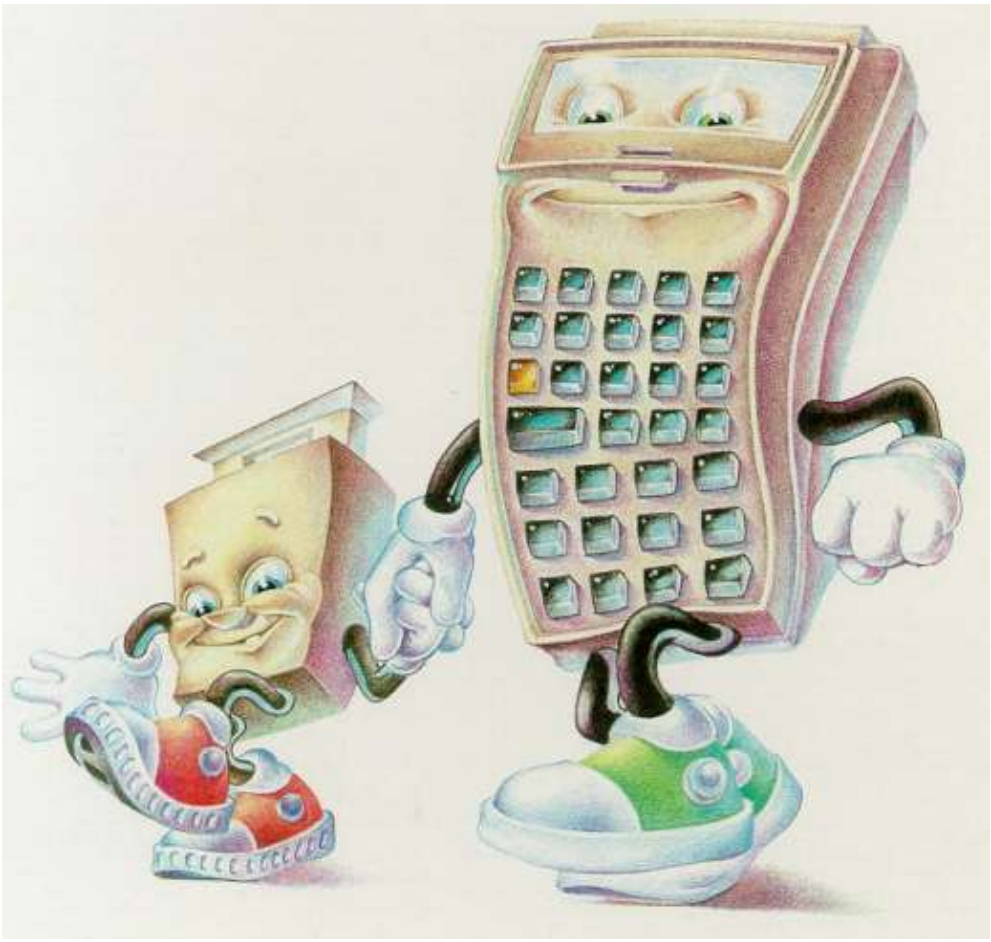
Function	Description	Input	Output	Author
POKER	Pokes Register	adr in Y, content in X	writes data to register	*W&W GmbH*
RAMED _	RAM Editor	RAM addr in X or current PC	Editor mode activated	*Håkan Thörngren*
READPG	Read Page	FileName in Alpha, pg# in X-reg	reads ROM image from HP-IL	*Raymond del Tondo*
READXM	Read All X-Memory	FileName in Alpha	reads X-Mem from HP-IL	*Skwid*
RENMFL	Rename X-Mem file	"OldName,NewName" in Alpha	file renamed	*Ángel Martin*
RETPFL	Retype X-Mem file	FileName in Alpha, type in X	file type changed	*Ángel Martin*
ROM? _ _	ROM Info	Prompts for XROM id#	pag# in X, #fcns. In Y	*W&W GmbH*
SAVEBF	Save Buffer	FileName in ALPHA, buff# in X	Saves buffer to X-Mem file	*Håkan Thörngren*
SAVEKA	Save Key-assignments	FileName in ALPHA	Saves KA to X-Mem file	*Håkan Thörngren*
SUMPG _	Sum Page	prompts for page#	Calculates & writes Chksum	*George Ioannou*
TGPRV _	Toggle Provate status	Prompts for prg name	status changed	*Sebastian Toelg*
WRTPG	Write Page	FileName in Alpha, pg# in X-reg	Rom image written to HP-IL	*Raymond del Tondo*
WRTXM	Write All X-Memory	FileName in Alpha	X-Memory written to HP-IL	*Skwid*
XQ>XR	XEQ to XROM	Program name in Alpha	Converts XEQ to XROM	*W&W GmbH*

Original authors are listed (to the best of my knowledge). W&W is credited for the CCD functions; let me know if you know the names of the actual programmers, which should include Holger Adelmann and W. Baltes according to the introduction in the CCD Module manual. With 22 functions from the CCD (in addition to the other system extensions) the initial spirit is preserved as a genuine derivative of the original masterpiece.

No doubt you've also noticed some redundancy with the TOOLBOX and RAMPAGE modules – it's true that several functions are "repeated" amongst these three, but as a general theme the OS/X should provide access to the most frequently used functions, whilst the other two would extend the toolsets with more specialized functions on the subjects that they cover.

The last remark is regarding the CX dependency: the AMC_OS/X is designed for the CX version of the 41 OS, as it profusely uses subroutines from the CX OS code. This was a compromise to maximize the functionality and the economy of ROM space - avoided having to replicate large code streams already available on the CX.

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



Note: Make sure that revision “L” (or higher) of the Library#4 is installed.

2. System Extensions.

The functionality described below is always available: no functions need to be executed and USER mode is not required. The implementation is based on Polling-point interrupts, no doubt the most appropriate approach to modify the standard behavior of the OS (akin to a sub-classing scheme).

2.1.- ENHANCED CATALOGS

Without a doubt, one of the most useful enhancements from the CCD was the extended CATalogs. Not only they preceded the CX implementation of EMDIR and ALMCAT, but also provided real system-wide extensions to enumerate HP-IL Peripherals [CAT-0], or files on a Mass Storage device [CAT-7] to mention just those two. Also useful to the point of addiction are the shortcuts for page catalogs 8-F – I for one can't use a 41 without them!

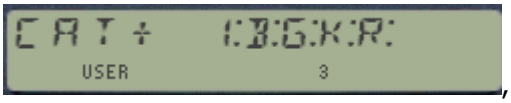
The following paragraphs are taken from the CCD Module manual – added here for completeness and your convenience. The additional functionality from the AMC_OS/X is also marked appropriately within each section.

The Catalogs

The original catalogs in the standard machine (six on the CX) are expanded to 16, and their functionality is considerably enhanced. All of the new catalogs may be halted during execution by R/S, and subsequently stepped through in either direction using SST or BST. In contrast to the operation of the native catalogs of the HP-41, the SHIFT annunciator remains lit during the use of BST (or SST). The key sequence SHIFT, R/S will even cause the catalog listing to be run in reverse. A running catalog may be speeded up by pressing any key other than R/S or ON. Pressing the back arrow key terminates the stopped catalog. The catalogs will now be individually described.

- **CAT"0** shows the ID or AID of all devices in the HP-Interface Loop if any are present. When the catalog is stopped the displayed device can be selected by pressing ENTER. If you press [**C**] and the selected device is selected, a Selected Device Clear (SDC) message will be sent to that device. When there is no HP-IL module in the calculator, the message 'NO HPIL" is displayed.

- **CAT"1** is a back-door entry for three different catalogs related to RAM and ROM areas. It presents a second menu of choices, as shown in the screen below:



, where:

New functionality:-

- "B" executes **BFCAT**, the I/O Buffer Catalog – explained later in the manual
- "G" executes **PGCAT**, the I/O Port Catalog, explained in section 2.6
- "K" executes **CHKSYS**, the system check function (see section 4.1)
- "R" executes **ROMLST**, showing all plugged XROM id's as alpha string
- "1" executes the normal **CAT 1** function, with no enhancements in the manner of execution. More information can be found in the HP-41 Handbook.

- **CAT"2**.- This catalog is greatly enhanced in its operation in comparison to the standard CAT 2 of the HP-41. When it is first executed only the "headers" of each ROM are displayed (like the HP-41CX). If the catalog is halted with R/S the user may press ENTER to view the function block of the currently displayed "header". When the desired function is located, it may be executed directly from the catalog by pressing XEQ (the function will be inserted in program mode), or the function may be assigned to a key by pushing the [**A**] key. A second press of the ENTER key returns you to the catalog listing of only ROM "headers".

  Another difference with the CX implementation is that the enumeration of the CX-Functions and Time Module follow the actual page order, thus the very first ROM listed is always the extended functions – not the last one as it's done on the "native" CAT 2.

- **CAT"3** has been extended with a 3-digit prompt. Use it to define the function to start from; choices are from 0 to 115 (in decimal). I adapted this feature from Poul Kaarup's **CAT3** function.

- **CAT"4**.- Like the function EMDIR of the Extended functions module and CAT 4 of the HP-41CX, CAT"4 displays the names, lengths and types of all files in extended memory. It has the additional feature of displaying the three additional file types used by the CCD Module. The three file types are: I/O Buffers (displayed as "B"), Matrices ("M'"), and key assignments files ("K"). If no extended memory is present the error message "NO XF/M" is displayed.

New functionality:-

The AMC_OS/X had added new tricks to CAT"4, the Extended Memory Catalog – namely completing the information about non-standard files (Matrix, Buffer and KA, missing on the CCD_OS/X version), plus adding five new file types to the list. The new file types are as follows:

File type	Mnemonic	File Type Id#	Used by	In Modules
Status Registers	**"T"**	07	SAVEST / GETST	RAMPAGE / POWERCL
Complex Stack	**"Z"**	08	SAVEZS / GETZS	RAMPAGE / POWERCL
Unassigned	**"Y"**	09	Future use	n/a
Unassigned	**"X"**	09	Future use	n/a
Hepax Data	**"H"**	10	Future use	n/a

Obviously the usefulness of additional file types is determined by actual functions that make use of them. This is the case for the Status registers and the Complex Stack types, with dedicated functions to save and restore the corresponding memory areas from/to X-Memory.

Note that the other types can still be used by means of the **RETPFL** function described elsewhere in this manual. While its argument can be any number between 1 and 99, having recognizable mnemonics in the catalog enumeration is a very useful feature.

- **CAT"5.**- Executes the function ALMCAT of the TIME module. When there is no TIME module in the calculator the message "NO TIMER" is displayed.

- **CAT"6**.- This catalog shows all key assignments in keycode order, starting at the Σ+ key and working its way horizontally and then dropping down to the next row. On the right side you will see the keycode and on the left side the function name will be displayed. Even synthetic key assignments (like "RCL M", or "TEXT 7") are shown correctly and not as an XROM number. Pressing [**C**] deletes the shown key assignment when the catalog is stopped. When there are no key assignments the message "NO KEYS" is generated.

- **CAT"7**.- Executes the function DIR of the HP-IL module. For a detailed description of this function see the owner's handbook for the HP-IL module. When this is not present in the calculator the message "NO HPIL" is shown.

- **CAT"8** to **CAT"F**.- These catalogs operate in a manner similar to the enhanced CAT"2 function of the module, except that each of these addresses a single page of the I/O ports of the HP-41. Both the catalogs and the ROM pages are numbered from 8 to F. As one might therefore expect, each of the catalogs in this group has a number identical to the ROM page whose content it examines.

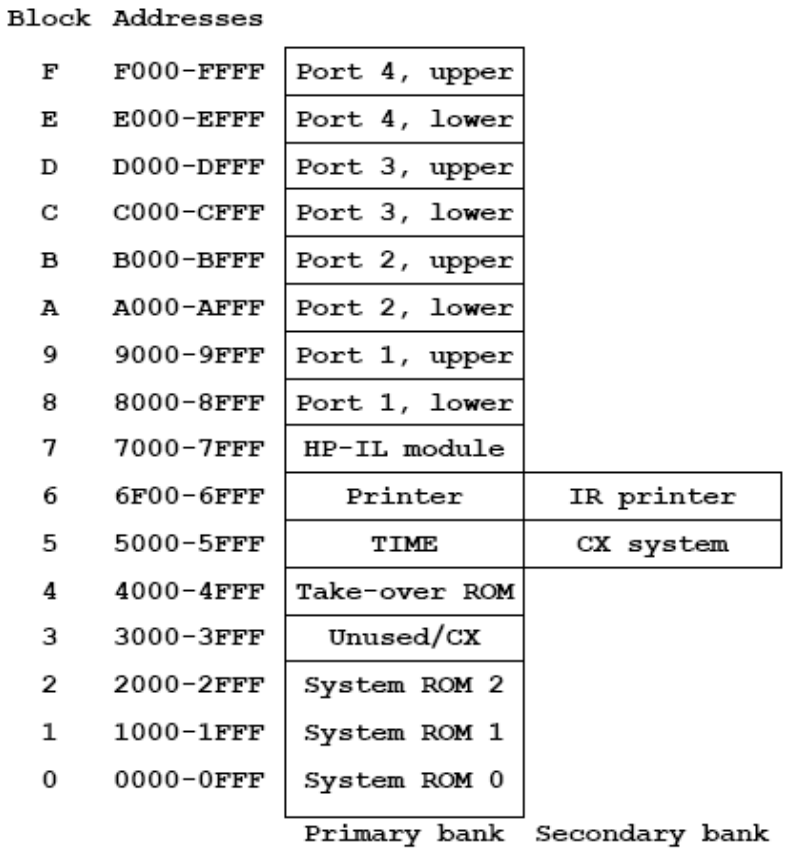
Each port of the HP-41 can be occupied by up to 8 Kbytes of program material (in non- bank switched configuration; or up to 32k if bank-switching is used to its max). Since most application modules address the lower 4K of the port they're plugged into, then the upper page of that I/O port is inaccessible under normal circumstances, and the corresponding catalog will display the message "NO ROM" for that address block. Some modules use both pages but have only one Function Address Table (FAT). In those cases the message "NO FAT" is shown as appropriate.

Note that there are no direct shortcuts to list the ROMs plugged in the "internal" pages of the I/O bus (pages 1 to 7). Pages 0 to 5 are used by the operating system, CX-Functions and TIME module code, therefore in practical terms only pages 6 and 7 are not covered. They have to be listed using the general CAT"2, stopping the listing at the corresponding ROM header.

Related functionality:-

Another function in the OS/X Module is **PGCAT**, taken from the HEPAX Module 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.

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:



Lastly but by no means least, the Buffer Catalog **BFCAT** completes the enhanced catalogs set – providing a simple and convenient way to list those buffers configured in the system. **BFCAT** behaves in every way identical to the other catalogs, with automated or manual enumeration using SST/BST, and with hot keys to delete and decode the buffer header. It'll be described later on in the manual.

Full House Configuration of the I/O Pages.-:

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

2.2.- EXTENDED XEQ/ASN

To understand the following you should have at least a basic knowledge about synthetic programming, or be familiar with the concepts involved in advanced programming.-

ASN

The enhanced ASN function permits the following keyboard entries:

a) The normal ASN function: if the user presses ASN followed by ALPHA, the standard ASN function of the HP-41 will be run.

b) The assignment of any two-byte function: when you press the ASN key with the OS/X Module present you will see the following prompt: "ASN: _ _: _ _". The calculator is prompting for two decimal byte values. When you key in two bytes and press any key after that the two-byte function is assigned to that key. If you first press the [**H**] key the operating system will prompt for hexadecimal values. With ENTER or the radix key [**;**] you return to the decimal prompt.

c) Assigning an XROM number: the ASN function of the OS/X Module allows the assignment of XROM numbers without the module with that XROM plugged in the calculator. After you have pressed ASN you simply press XEQ and you'll see "ASN XROM:_ _". This prompt initially requests input of the ROM id# number (i.e. the portion of the XROM number that precedes the comma, such as 05 for the OS/X module). After the entry of these digits, the prompt becomes "ASN XROM:05:_ _", which indicates that input of the function id# - that is to say the portion of the XROM number that follows the comma - is now expected (e.g. 01 for the function ABSP of the OS/X Module). The prompt becomes "XROM:05:01 _", which requests input of the code for the key this function is to be assigned to.

Note that pressing back arrow at this point does not cancel the action, but assigns the function to the back arrow key instead. You should manually un-assign it if that was unintentionally made, using the key sequence: ASN, ALPHA, ALPHA, key.

Related functionality:-

Note that here we've used the function XROM id# to make the assignment, and that it is restricted to two-byte functions. Another function in the OS/X module is **ASG**, which features a more capable approach using the function names instead, for additional convenience and ease of use. **ASG** was taken from the ML ROM, compiled by Fritz Ferwerda from the Dutch PPC Chapter.

XEQ

The enhanced XEQ function allows the following keyboard entries:

a) The normal XEQ function: If ALPHA or a number is pressed after XEQ, we obtain the normal XEQ function. It is the same as the standard XEQ on the HP-41.

b) The execution of any two-byte function: When you press XEQ and then ENTER you will see the prompt "XEQ:_ _:_ _". The calculator is prompting for two decimal values. When you key in two values the function is executed or inserted into a program. If you press [**H**] before keying in any value you'll see "XEQ"_ _"_ _" and the calculator prompts for a hexadecimal input. By pressing ENTER or the radix key [**;**] you are returned to the decimal prompt.

c) The execution of an XROM number: the expanded XEQ function provided by the OS/X Module also permits the execution of function or application programs by their XROM number, even if the module is not present in the HP-41. The key sequence XEQ, ENTER, XEQ generates the prompt "XEQ XROM:_ "This initially prompts to input the ROM id# number (the portion of the XROM number that precedes the comma, such as 05 in the OS/X module). After the entry of these digits, the prompt becomes 'XEQ XROM:05:_ _" , which indicates that the input of the function id# - that is to say the portion of the XROM number that follows the comma – is now expected (e.g. 01 for the function **ABSP** of the OS/X Module). If the calculator is in RUN mode and the appropriate module is plugged in, the function is immediately executed; otherwise the error message "NONEXISTENT" is displayed. If the HP-41 is in PRGM mode then the instruction is inserted as a program line.

Note: to avoid confusion, throughout this manual the appearance of the colon (:) preceding an input prompt indicates that the number to be input is of the decimal form; whereas if the colon is replaced by quotes (") the input is expected in hexadecimal form.

Example:- Assign the synthetic function 'RCL IND e" to the LN key.

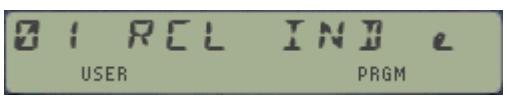
A quick look into the byte table determines that the byte values required are in Hex 90,FF and the key code is 15. Armed with that information it's easy to just fill the prompts in the OX/S ASN function:

ASN, [**H**], 90, FF, 15

Alternatively you can execute the **ASG** function and spell out the function name. Note that the ALPHA mode is turned on automatically for this:

ASG, R, C, L, space, I, e (lower case)

In either case executing CAT"6 will show the function assigned with its correct name:

and in PRGM:


2.3.- DIRECT MEMORY FUNCTIONS

To simplify the insertion of synthetic program lines, the OS/X Module provides the capability for the direct entry of synthetic instructions. All memory access functions (RCL, STO, X<>) can now be accessed directly from the keyboard and use to address all of the status registers of the HP-41. Thus access to and manipulation of the contents of registers {M, N, O, P, Q, a, b, c, d, e, +} is now no more complex than working with {X, Y, Z, T, L}.

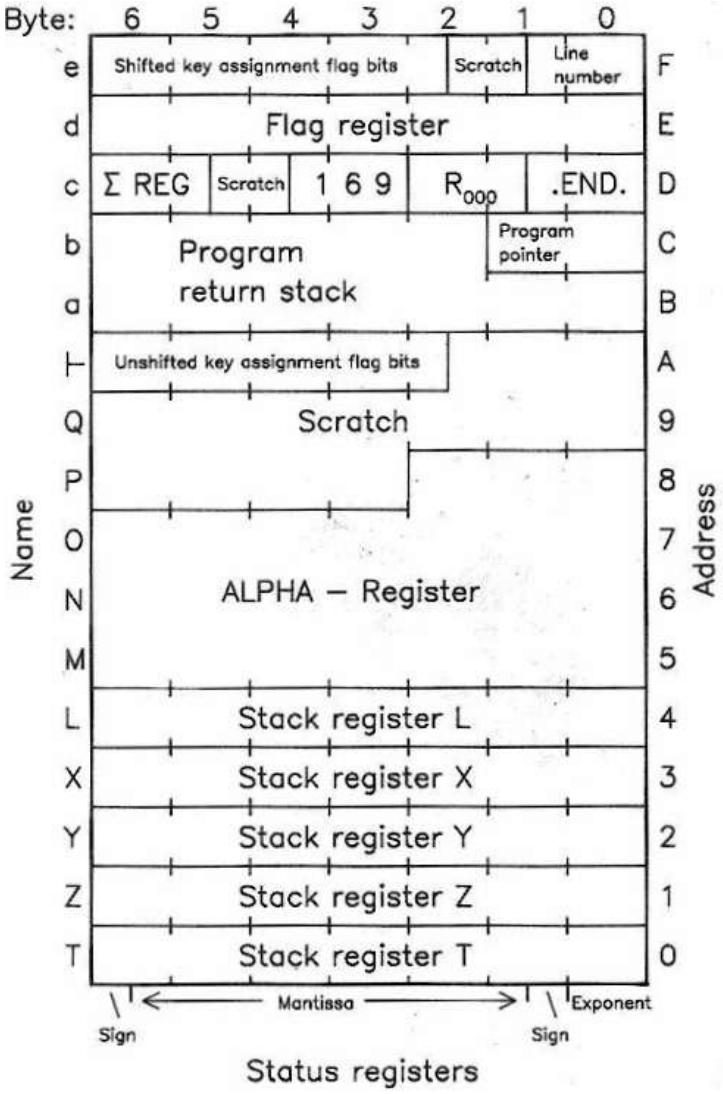
This is valid for both the direct and INDirect modes of the functions. For instance, the keystroke sequences used to apply these functions to the status register "d" would be:

RCL, [.], [D]
RCL, SHIFT, [.], [D]

Exercise caution in manipulating status register contents: Altering the contents of registers "+" and "a" though "e" can lead to a MEMORY LOST condition or to a system crash if the register contents are improperly altered.

Alteration of the "cold start constant" 169 in register "c" will always result in MEMORY LOST. Before experimenting with these registers the user should thoroughly familiarize himself with the theory and practical applications of synthetic programming.

Even more interesting considerations apply to the utilization of status registers during program execution. Remember that register "b" holds the current program pointer, i.e. it's a powerful way to jump to other programs, or even ROM space without any global label.

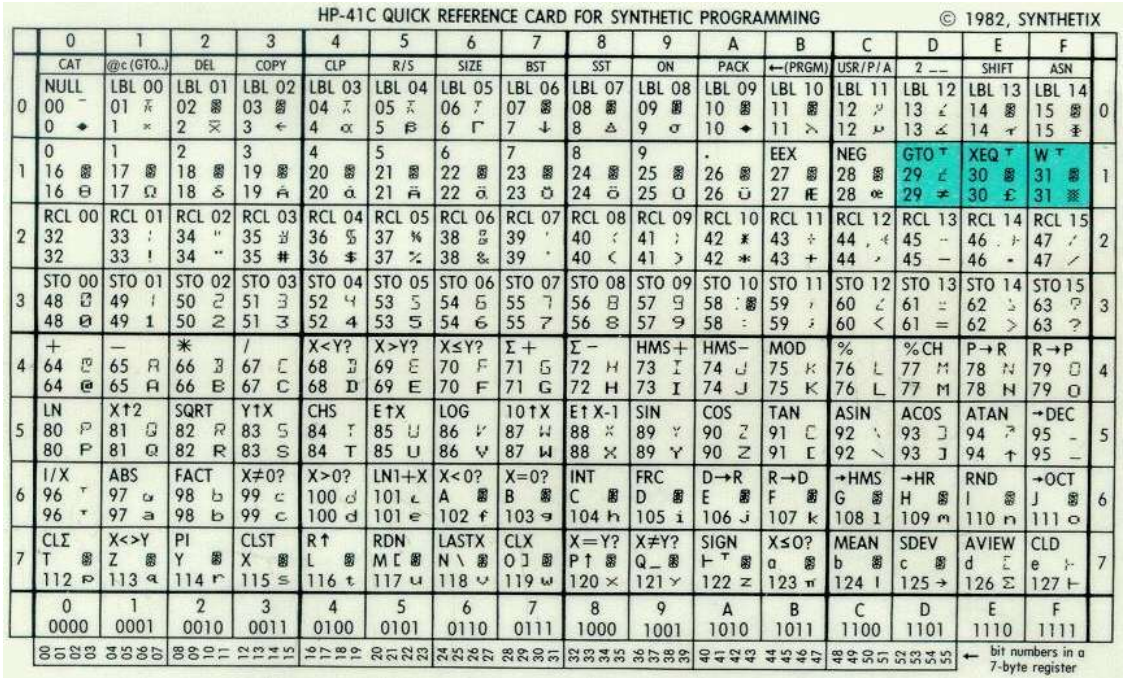


Related functionality:-

The functions **SAVEST** and **GETST** in the RAMPAGE Module can be used to save/restore the complete set of status registers in extended memory. Make sure you understand the implications of a "hot-swap" of the status registers set before doing it!

2.4.- ALPHA CHARACTERS INPUT

The OS/X Module enables the user to place in the ALPHA register, or to enter directly into a program line, any of the 256 character bytes available on the HP-41 (see byte table below). This was only available previously using the X-Functions module or synthetic programming techniques. Exploiting the use of direct entry of lower case and special characters presents lots of possibilities, especially to programmers making extensive use of printers and HP-IL peripherals. It is readily apparent that the direct entry of lower case and special characters greatly facilities the ease of use and "byte economy" in programming.



The lower case and special character entry mode of the OS/X Module is available in ALPHA mode when USER mode is off. The special keyboard overlay companion to the CCD Module has the printer-control and other special characters listed according to the color code described in the table below:

	USER Mode ON	**Normal Alpha keyboard**
	Blue Letters on key faces	Capital letters A-Z and some special chars
	Blue Letters on Overlay	Special chars available by pressing SHIFT
	USER Mode OFF	**Lower case mode active**
	Blue Letters on key faces	Lower-case letters A-Z
	Red letters on overlay	Special chars for un-shifted keys
	Green letters on Overlay	Special chars available by pressing SHIFT

If the desired character is not available on the keyboard you can key it in using its decimal or hexadecimal value by pressing SHIFT ENTER for the decimal, or press SHIFT ENTER, [**H**] if you want to enter it in hex. Note that it is not possible to cancel this byte prompt. If you see the prompt, just key in any non-zero character and delete it afterwards.

When a special char is also a printer or IL device control code it is shown on the overlay on the right side of the key with its control code function name. If the lower case mode is not desired it can be suspended using the function **TGLC** – which toggles its status ON/OFF upon each repeated execution.

2.5.- PROMPT LENGTHENER

This functionality is borrowed from the ML ROM – also using the polling points technique although in a simpler implementation.

While executing a prompting function, pressing the [ON] key will lengthen the prompt to enter more digits. This can be used for arguments exceeding the standard prompt length, such as RCL 101 (shown as RCL 01) to RCL 111 (shown as RCL J), or as a handy shortcut for synthetic ones.

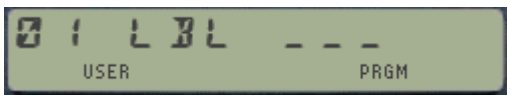
To enter RCL M for example: RCL [ON], the display shows "RCL _ _ _", enter 117 and the function will be executed or inserted in the current program (in PRGM mode).

The table below shows the correspondence between the extended arguments and the actual registers used. Note that:

- In the range 102-111 the display is showing the conventions used for the LBL instructions, but the actual registers are correct.
- In the range 112-127 the registers used are the status registers instead of memory data registers. This is what we take advantage of to key in status registers as arguments.
- From 128 and up the instruction changes to indirect indexing. This is due to the way indirect addresses are built by the OS; adding hex 0x80 to the register number. As a consequence of this and the previous point, for registers 112 and up only the standard indirect addressing is available to store and recall their contents.

	Argument	Shown:	Argument	Shown:	Argument	Shown:
	100	00	112	T	124	b
	101	01	113	Z	125	c
	102	A	114	Y	126	d
	103	B	115	X	127	e
	104	C	116	L	128	IND 00
	105	D	117	M	129	IND 01
	106	E	118	N	130	IND 02
	107	F	119	O	131	IND 03
	108	G	120	P	132	IND 04
	109	H	121	Q	133	IND 05
	110	I	122	|-	134	IND 06
	111	J	123	a	135	IND 07

The prompt lengthener is meant to be used with the following functions: STO, RCL, X<>, LBL, XEQ, and GTO. It will however also be triggered during other prompts (like SF, CF, FS?, CAT) which obviously have no practical application for it, and that will typically generate the NONEXISTENT error message.



2.6- ADDITIONAL CATALOGS

CATalog functions are notoriously complex and take up a significant amount of space – but if you're like me you'll like to have good visibility into your machine's configuration. Therefore you'd hopefully agree with me that the usability enhancements they provide make them worthwhile the admission price.

2.6.1. Buffer Catalog.

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

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.

HP-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 exit – which thankfully they don't.

For instance, plug the OS/X 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 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	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 Buffer
14	Advantage, SandMath	INTEG & SOLVE scratch
15	Mainframe	Key Assignments

The id# 15 is not really a buffer type, but reserved for the key assignment registers.

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.

2.6.2. I/O Page Catalog.

Note that although these functions are not accessed using the system extensions described before, given their system scope it only feels appropriate to include them in this section of the manual.

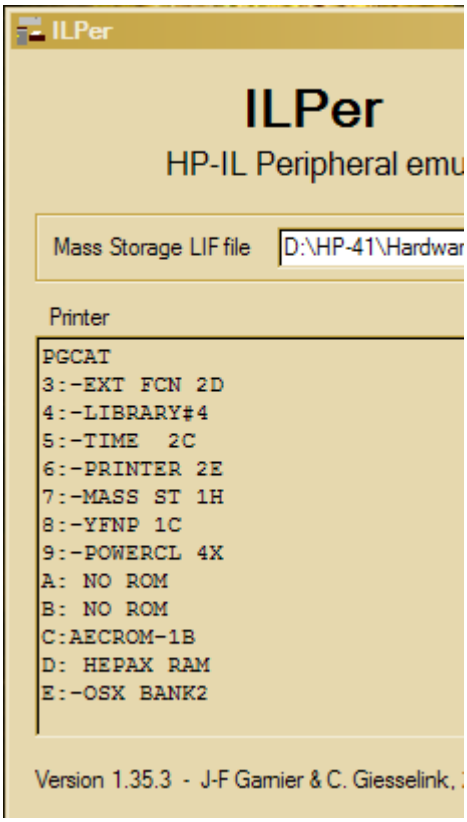
PGCAT	Page Catalog	VM Electronics	Source: HEPAX Module

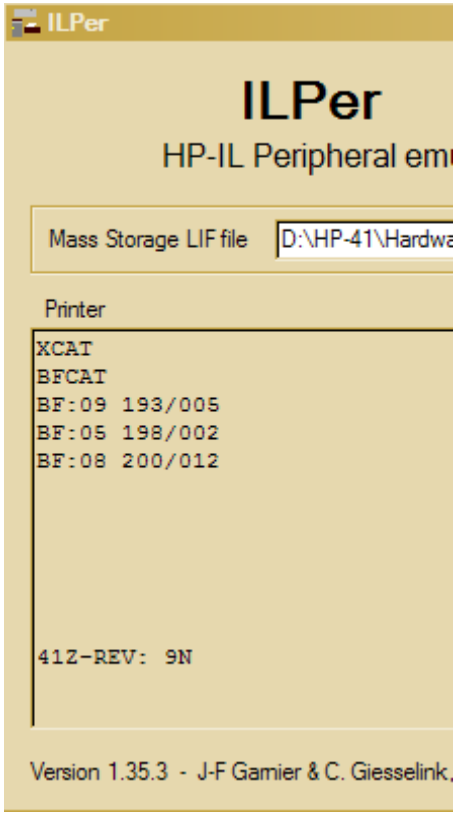
PGCAT Lists the first function of every ROM block (i.e. Page), starting with Page 3 in the 41 CX or Page 5 in the other models (C/CV). The listing will be printed if a printer is connected and user flag 15 is enabled.

- 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:





3. RAM EDITOR

Adding a second bank to the OS/X Module provided a large amount of space for additional functions to be included. The choice of functions added over previous incarnations was clearly meant to have a comprehensive and self-contained function set that included some of the best examples ever written for the HP-41 system. RAM editors are no doubt amongst these, and as such one is included in the bank-switched version of the module.

RAMED	**RAM Editor**	Uses GETKEY [KEYFNC]	Håkan Thörngren

Editing RAM memory with RAMED.

Written by MCODE master 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. A real tour-de-force and a masterful implementation without any doubt.

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

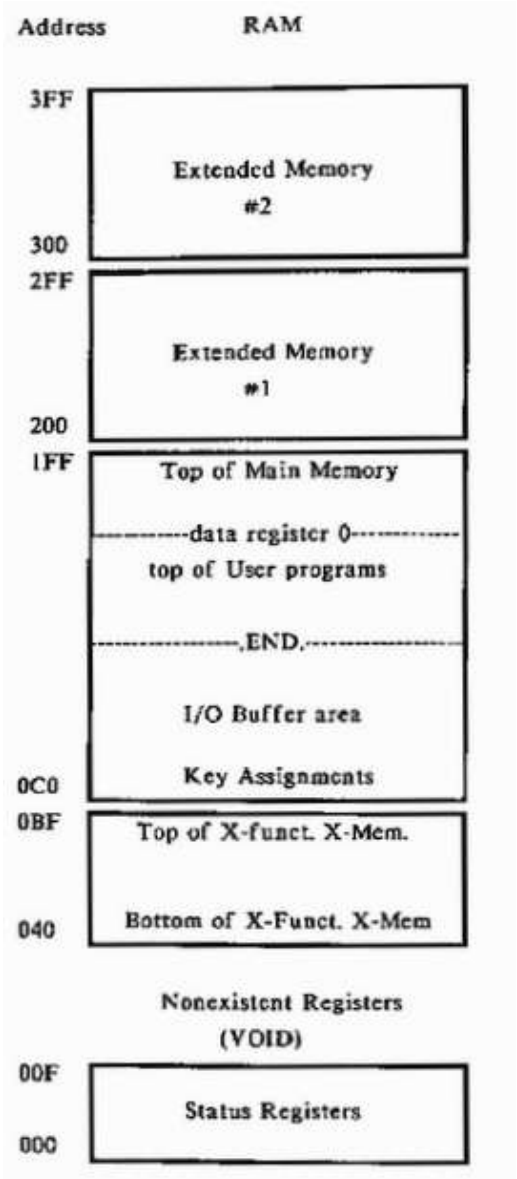
A few last remarks are in order:

- **RAMED** is a very powerful tool: the contents of all memory can be edited, including the Status Registers, I/O Buffers, KA registers, and of course X-Memory files (see memory map below). Be very careful not to alter the contents of those system registers inappropriately to avoid MEMORY LOST or system crashes.

- **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.

- **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** is also included in the RAMPAGE module, named **RAMEDIT** there **–** because the ZENROM function RAMED is also included in the module.



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

4. SYSTEM & PAGES

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

4.1. The system as a whole.

CHKSYS	**Checks ROMS plugged in**		Ángel Martin
ROMLST	**Lists ROMS plugged in**		Ángel Martin
OSREV	**Shows OS Revisions**	Under **PGSIG**	Nelson F. Crowle

- **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:

- **OSREV** is now "hidden" under **PGSIG**, used with any argument larger than 15 (the highest page number on the I/O Bus). OSREV shows the revisions for the three first pages, containing the core Operating System code (in ROMS 1/2/3) / which for an unmodified HP-41CX are as follows:

- **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?

Note that due to the limited number of FAT entries available, on the OS/X Module this function is "hidden" under the section header "-OS/X BANK2". You can use its XROM id# to assign it, execute it or to insert it as a program line: XROM 05,36

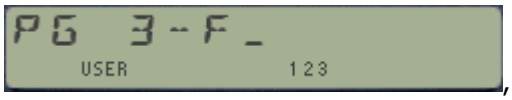
4.2 The Pages within.

SUMPG _	**Sums Page checksum**	Page# in prompt/X	George Ioannou
PG? _	**Page vital constants**	Page# in prompt/X	W&W GmbH
PGSIG _ _	**Page Signature**	Page# in prompt	Ángel Martin
ROM? _ _	**Rom vital constants**	XROM id# in X	W&W GmbH
CHKROM _ _	**Verifies ROM checksum**	XROM id# in prompt	HP Co.
READPG	**Reads page from HP-IL**	Page# and FileName	R. del Tondo (?)
WRTPG	**Writes page to HP-IL**	Page# and FileName	R. del Tondo (?)

- **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 OS/X page it'll return the vital constants for bank-2 (where its code resides), which strangely enough are 05,10 in X (as it was explained at the beginning of the manual)

  Considerable trickery has been used modifying this function 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.

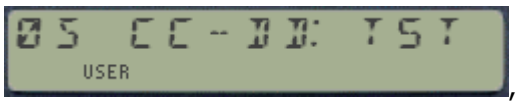
- **PGSIG** will retrieve the signature string of the ROM plugged in the page entered at the prompt (in decimal format) – or in the X register if used in a program. If no ROM is plugged it'll return four "@" characters.

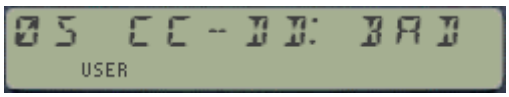
input prompt

represents a "blank" signature value.

Entering any value greater than 15 will trigger the **OSREV** function instead, as described before.

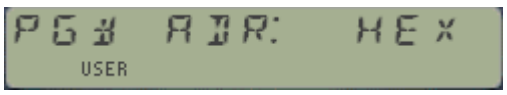
- **CHKROM** will check the ROM which XROM id# is input at the prompt (or in X when run in PRGM mode) for the correct checksum byte value. The display shows information message while the test takes place, followed by a confirmation or a warning depending on the case.

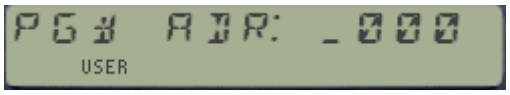




  Incidentally it's more than likely that if you run **CHKROM** on the OS/X itself the result is "BAD". This is not because of an error; I just usually don't bother to update the checksum values, as the code is updated very frequently.

- **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.





- **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.

  Note that the file formats on disk will be compatible with the HEPAX functions that perform the same tasks, but not so with equivalent functions from the ML ROM, Eramco MLDL or other EPROMS from the Dutch PPC Chapters. There are thus two "standards" that cannot be intermixed.

5. ALPHA & DISPLAY UTILS

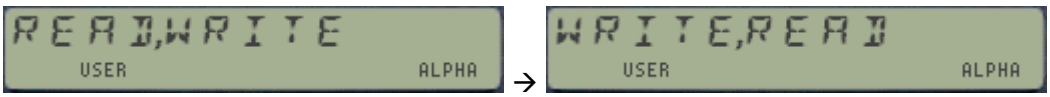
The OS/X Module has a small set of ALPHA functions, chosen for being in the original CCD Module – also complemented by a few others of general applicability. For a comprehensive list of Alpha and Display related functions you should use the ALPHA_ROM (or the POWER_CL), which contains pretty much all you would ever need.

ABSP	**Swap Alpha and Regs.**	1st. RG# in prompt	Ken Emery (original)
ARCLI	**Append Integer part**	Takes absolute value	W&W GmbH
ASWAP	**Alpha Swap**	A,B in ALPHA	Ángel Martin
CLA-	**Clear chrs after blank**	Text string in ALPHA	W&W GmbH
DTOA	**Display to Alpha**	Opposite to AVIEW	Ángel Martin
DTST	**Display Test**	Source: PPCJ V18 N8 p14	Chris L. Dennis
PMTA _	**Alpha Prompt**	Prompts for ALPHA text	W&W GmbH
TGLC	**Toggle Lower Case**	Toggles the LC mode flag	Ángel Martin
MSGE	**Displays OS Message**	Msg. id# in prompt	Nelson F. Crowle

- **ARCLI** 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). Very similar to **AIP** in the Advantage Pac, or **AINT** in the SandMath and Alpha ROM.

- **ABSP** removes the rightmost character from the ALPHA register. It is equivalent to switching ALPHA on, pressing the back arrow, and switching it off again.

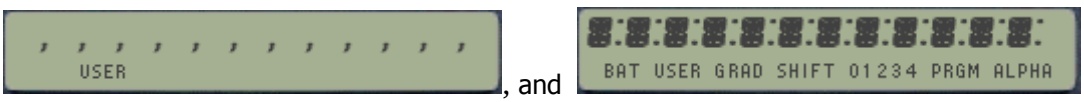
- **ASWAP** swaps the strings at the left and right of the comma character. Very handy for X-Functions data input. Does nothing if comma is not there. For example:



- **CLA-** deletes all characters in ALPHA from the right, until it finds a blank space. The blank space is not erased. If the ALPHA register is empty or contains only blank spaces or letters, the whole contents will be erased.

- **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.



- **PMTA** is one of the trademark functions of the original CCD Module. It prompts for ALPHA text, using the existing content in ALPHA as prompt cue – or adding 'TEXT: " if ALPHA is blank. Like the mainframe function PROMPT, it stops the program execution when used in a program. But let's read the original description from the CCD Manual:

  Description.- The function **PMTA** gives the possibility of a comfortable ALPHA input. It consists of two parts:- If executed in a running program, the program ruin will be interrupted and the program pointer will be set back by one program line to the function PMTA. Now the OX/S Module input flag (bit 5 of byte 4 in register c(13) is set, the ALPHA register is switched on and a underscore sign is placed into the display. Using R/S and ON the function can be terminated, without the loss of the original ALPHA contents.

  If a different key is pressed, all of the previous contents of the ALPHA register are erased, which has no influence on the shown indication. If the depressed key is a letter key, the ALPHA register will be overwritten with the corresponding letter and will be appended to the display. After pressing the R/S key, **PMTA** is executed for the second time. The function recognizes this by the fact that the input flag is still set. The flag is now cleared, ALPHA is turned off and flag 23 is ser if there was any input into ALPHA.

  Further hints.- If during execution of PMTA the ALPHA register is empty, the string "TEXT: " is shown. Like all prompt functions in the module, PMTA executes BST once during the execution of the program. If BST is executed at the end of a a large program it will take quite a long time – therefore it is advisable to put all prompt functions at the beginning of the program (possibly in a subroutine), or shortly behind a global label.

- **TGLC** toggles the status of the lower-case mode flag (bit 16 in status register c). Use it when you want the lower-case to be disabled during ALPHA input, even in USER mode. Note that the OS/X Module intentionally uses the reversed convention for activation of the lower case mode: USER must be ON, not OFF as it is the case in the original CCD Module.

  Remember that most of the lower case characters will be shown as starburst in the HP-41 display, but will be properly interpreted by the printers or other HP-IL peripherals.

- **MSGE** is a nice little routine that piggybacks on the OS routine [MSGE]. Use it to display standard OS messages - totally or partially – by appropriately choosing the message index. This provides an obvious byte count reduction in a FOCAL program. The table below shows the arguments for the complete messages.

MSGE#	TEXT
10	A:
11	A:K
14	A,Z.
24	ALPHA DATA
34	DATA ERROR
45	MEMORY LOST
56	NONEXISTENT
59	NULL
67	PRIVATE
79	OUT OF RANGE
86	PACKING
95	TRY AGAIN
98	YES
100	NO
103	RAM
106	ROM

6. HEX FUNCTIONS

This section includes several of the Binary/Hex functions from the original CCD – but not the complete set. Here again a compromise had to be made in order to make it all fit in a single FAT. Besides de CCD, the DIGITPAC is another module that has a more complete set of Binary/Hex functions, including the HEX, Oct, Bin conversions and viewers from the HP-IL Devel Module and Advantage Pac.

6.1 The Hexadecimal Number System

The base of this number system is 16(d). The numbers 0-9 as well as the letters A to F are used to represent 11(d) to 15(d). The carry over to the next place occurs at 16(d). The table below shows the correspondence between binary, decimal and hexadecimal numbers up to 15(d):

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

As you can see on the table, Hex numbers are the natural choice to represent four binary digits (4 bits); therefore the hexadecimal number system uses the 4 bits of a nibble in the full value range. Each nibble can represent values from bin 0000 to 1111, of in hex from 0 to F. This number system is often used in computer science.

Representation of negative Numbers

When using a limited count of digits m to represent numbers, the value range to the numbers to be represented n is $0 < n < 2^m$, meaning that the greatest number will be $2^m - 1$. Note that no negative numbers are included in this system of notation. To remove this deficiency, the concept of "complement" is introduced.

The Complement (from a reference K)

The K-complement of a number x is defined as the difference from K of that number: Com(x) = K–x for which k is fixed by the chosen complement. Since in the binary number system the usual values of K are 2^m, and $(2^m - 1)$, we usually speak of "one's complement" or the "two's complement" – representing the difference from either the greatest number represented or the one after it. In general, for any base system "b" there will be a "b" and a "b-1" complement – like the 9's and 10's complement in the decimal system where b=10.

Distribution of ranges of values.

If there seems no reason yet for the existence of complement notation, remember that to this point we haven't dealt with negative numbers. We shall include these by adopting an arbitrary distribution of value ranges in our number systems. For instance in the binary system, we'll define a negative number as one which most significant (leftmost) bit is set (has a value of 1). Arranging number distributions and using complements change the ranges of values as follows:

No Complement	$0 < x <= (b^m - 1)$
b-Complement	$-b^{(m-1)} <= x <= b^{(m-1)} - 1$
(b-1) Complement	$-b^{(m-1)} - 1 <= x <= b^{(m-1)} - 1$

Basically in the signed modes (with complement) we divide the original total value range b^m-1 in two half sections, allocating one of them for negative numbers and the left over for positive.

Complement (Signed) Modes.

The transformation of decimal numbers represented in the binary system (and vice-versa) varies with the signed mode used. The three modes are:

- Unsigned mode (no complement, only positive numbers)
- 1's complement mode
- 2's complement mode

The 1's Complement of a number is calculated by subtracting this number from the greatest representable number in the chosen word size (number of bits). For instance, say the word size is 5 bits – then the 1's complement of a number "a" is (11111 – a). The computer simply inverts all bits of the original number, i.e. executes the logical function "NOT". Through this segmentation of the initial value range (arbitrary but specific), all negative numbers have their highmost bit set, which plays the role of the minus sign. In the q's complement mode the number of positive and negative numbers represented are the same, i.e. even zero has two possible representations: +0 and –0, which in binary would be 0000 and 1111 (always assuming a word size of 5 bits).

The 2's Complement of a number is calculated by adding one to its 1's Complement. Using the same example of word size 5 bits, the 2's Complement of a number "a" would be (11111 – a) + 1, The 2's complement of bin 10001 is the number bin 01111. Notice how also in this segmentation the leftmost bit is set for all negative numbers, and therefore takes over the role of the minus sign. In the 2's complement mode there is one more negative number than in the positive number range, since zero only has the representation +0

The Unsigned Mode (no complement). Since the Complement mode employs one bit as the negative sign, the range of values for a word size of 8 bits in the 1's complement is from –127 (d) to +127(d), and in the 2's complement from –128(d) to +127(d). Note that in both cases those are 256 values. For cases when only the positive number range is needed, the precedence sign bit is not necessary (unsigned mode) and the value range is used for all cases from 0 to 255(d) - assuming the same word size of 8 bits.

In the table below you can see the correspondence between the three sign modes for decimal numbers 0-15 represented in binary, using a four-bit word size:

Dec	Hex	Binary	Unsigned	1's Compl.	2's Compl.
0	0	0000	0	0	0
1	1	0001	1	1	1
2	2	0010	2	2	2
3	3	0011	3	3	3
4	4	0100	4	4	4
5	5	0101	5	5	5
6	6	0110	6	6	6
7	7	0111	7	7	7
8	8	1000	8	-7	-8
9	9	1001	9	-6	-7
10	A	1010	10	-5	-6
11	B	1011	11	-4	-5
12	C	1100	12	-3	-4
13	D	1101	13	-2	-3
14	E	1110	14	-1	-2
15	F	1111	15	-0	-1

General Conventions used for Hex Functions

All Hex functions in the OS/X Module follow the same conventions, as follows:

1. The current word size is defined with the function **WSIZE**. Its value is stored in a dedicated Buffer (with id#-5), from where it's read in every required instance. The default setup (and the one after a MEMORY LOST condition) is word size = 8

2. The current sign mode. The decimal and the internal hex representations are linked by the complement, or sign mode employed. Before executing a certain arithmetic function the displays always shows the number in decimal – after having been converted to binary using the signed mode and back to decimal again. The default setup is unsigned mode.

3. The functions only use the integer part of the number in the X register.

6.2 Functions to manage Word size and Signed Mode.

WSIZE _ _	**Sets word size**	Value in Prompt	W&W GmbH
WSIZE?	**Recalls current word size**	Wsize to X-reg	Sebastian Toelg
CMP _	**Sets sign mode**	0,1,2 in prompt	W&W GmbH

The word size is set using function **WSIZE**. It sets the word size "*bb*" used by all hex functions in the OS/X Module. The allowed range is 1 to 32 bits, and its value is stored in the dedicated CCD buffer id#5. A value of zero clears the word size input in the I/O buffer, and sets the word size to its default value of 8 bits

WSIZE is now a prompting function; simply input the value using the numeric keys or the 2 top keys as shortcut for 0-9. When used in a program, the function's argument will be taken from the X register instead.

The argument (number in X) may be negative as well as contain digits after the decimal point, but only the absolute integer value is used. "DATA ERROR" will be shown when numbers outside the range –32 to 32 are used. "NONEXISTENT" will be displayed when numbers over 1,000 are used. The selected word size has no influence on the contents of the stack or memory data registers. Sometimes when executing **WSIZE**, the error message "NO ROOM" may occur, indicating there are no available memory registers to allocate the I/O buffer. In these cases more memory space needs to be made available before selecting the word size.

To find out the current word size set you can use function **WSIZE?**, It returns the value to the X-register, lifting the stack. The value is read from the I/O buffer, or defaulted to 8 if no buffer exists.

The signed mode is set using function **CMP** – which in the OS/X Module is also a prompting function, with valid arguments being 0, 1, and 2 only. Any other input will simply be ignored: no error message will shown, and the prompt will be maintained – which is a more efficient way to handle the error conditions. When used in a program the argument will be taken from the program line following **CMP** - a technique called "non-merged" functions, extensively used in other advanced modules like the HEPAX, SandMath, etc.

Note that **CMP** in the OS/X Module consolidates three functions from the original CCD Module into a single one (**UNS**, **1CMP**, **2CMP**). This has the advantage of freeing up two FAT entries, the most limiting factor in this case where ROM space is not at a premium anymore thanks to the bank-switching design. Note also that the current sign mode information is not saved in the CCD I/O Buffer, but instead it is stored in the scratch area of status register c(13). More about this later in the manual.

Selecting a sign mode replaces the previous one if set. The contents of the X register are put in the LASTX register and then replaced by the new representation of the hex number, i.e. the number in X is first converted to binary using the old sign mode and after switching to the new mode is converted back into a decimal number.

If the number in X cannot be represented in the new mode, (i.e. if it lays outside of the value range defined by the current word size) the error message "DATA ERROR" will occur and the mode will NOT be changed.

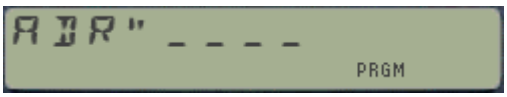
6.3 Input and Output Hex functions.

PMTH	**Hex Prompt**	Input in prompt	W&W GmbH
ARCLH	**ARCL X in Hex**	Appends value in X	W&W GmbH
VIEWH	**Views Hex equivalent**	Shows Hex value	W&W GmbH
XTOAH	**Appends hex-characters**	Appends hex bytes	W&W Gmbh

The function **PMTH** allows for the input if hex numbers. It writes the equivalent decimal value into the X register, and the stack is lifted before the value is copied to X. The number of digits ion the prompt is dependent on the current word size – taking one prompt per each 4-bits.

The function rejects input values too large for the current word size. Only keys 0-9 and A-F are active. The function can be terminated by pressing the back arrow or ON.

In program mode, text can be shown during the execution of **PMTH** to specify the data wanted for the prompt. For example, with "ADR" in ALPHA, and a current word size of 16, executing **PMTH** in a program will prompt for:



- The function **VIEWH** shows (views) the hexadecimal equivalent of the number in X – in accordance to the current sign mode and word size. The value must lie within the allowed range or otherwise a "DATA ERROR" message will be put up. Alpha data in X generates the "ALPHA DATA" error condition. For numbers with a hex representation with fewer digits than the current word size is set for, leading zeros are placed in the most significant digits.

- **ARCLH** is the ALPHA counterpart to the same functionality: it appends the hex equivalent of the number in the X register to the contents of the ALPHA register. The same considerations as above apply as to the allowed number range and error conditions.

- **XTOAH** appends one or more characters with the value of the X register converted to Hex to the contents of the ALPHA register. The value of the characters to be appended depends on the current word size and sign mode. It is therefore very similar to **XTOA** but in hex mode.

  Example: set the word size to 10, input 340 in X and execute **XTOAH** – Since 340(d) = 154(h), this will append two characters to ALPHA, byte 01 and byte 54

6.4 Random numbers revisited.

Not strictly related to the topic but included in this section nonetheless – as a surrogate for a dedicated Math coverage. The random number functions also use the CCD Buffer to store an initial seed, from which all random numbers are generated using the function RNDM.

- **SEED** furnishes an initial value for the computation of a random number using the function RNDM. The seed value is stored in the I/O buffer of the CCD Module. Only the fractional part of the X register is used for the starting value.

Note that with CLX, **SEED** you can clear the random number register from the buffer, thus freeing one more register for program storage. Furthermore, if the buffer only contains the seed information it will be removed altogether from the RAM I/O area. As a remainder, the buffer can also have the current word size, as well as possibly the current Matrix file name if you're also using array or Matrix functions from the CCD Module, the Advantage Pac or the SandMatrix Module.

- **RNDM** calculates the next random number, and replaces the seed with itself to be used in the next execution of the function. The random number has a value between 0 and 1. If there is not enough memory for the I/O buffer the message "NO ROOM" is displayed.

Note that the initial choice of seed is determinant of the sequence of subsequent random values, as they all follow the same expression:

RNDM = FRC(SEED * 9,821 + 0.211327)

So it all depends on the initial seed, so to speak. In this regard this implementation falls short of a true random character, which would require another system like the TIME module to pick up the beginning of the sequence in a truly random way – based on the current date and time, as an example.

Examples:- with a starting value of 0.1, the following random numbers are generated:

0.1, SEED
RNDM -> 0.311327
RNDM -> 0.753794
RNDM -> 0.222201
RNDM -> 0.447348 etc.

In case you wonder:- Word size format in the CCD I/O Buffer.

The word size format is stored in any of the three registers that buffer#5 can possibly have, regardless of which one – depending on when it's set and what was already configured. The module recognizes it by its first two nibbles, which are "F0" – as if it were a normal KA register. Then the actual word size is stored both in binary and hex formats; using nibbles 3 to 8 for the binary form and the S&X field for the hex equivalent.

The examples in the table below should clarify:

Word Size	Buffer Register
2	F0.000000003.002
4	F0.00000000F.004
8	F0.0000000FF.008
16	F0.00000FFFF.010
32	F0.0FFFFFFFF.020

7. SYSTEM UTILITIES

7.1. Main- and X-Memory Utilities.

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

CLEM	**Clear Extended Memory**	No inputs	Håkan Thörngren
RENMFL	**Rename File**	OLDName , NEWName	Ángel Martin
RETPFL	**Retype File**	NAME in Alpha, type in X	Ángel Martin
READXM	**Read X-Mem from Disk**	FNAME in Alpha	Skwid
WRITXM	**Write X-Mem to Disk**	Disk FNAME in Alpha	Skwid
PEEKR	**Reads register content**	Absolute address in X	W&W GmbH
POKER	**Writes register content**	Adr. In Y, NNN in X	W&W GmbH
PC<>RTN	**Exchanges PC and RTN**	Values exchanged	W&W GmbH
VRG _ _ _	**Decodes Register**	RG# address in X	Fritz Ferwerda

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

- **CLEM** is an expeditious Extended Memory eraser – all files will be irreversibly gone in the blink of an eye, 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.

- **RENMFL** 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 filed named NEWNAME already ("DUP FL" error message).

- **RETPFL** 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

  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

- **WRTXM** and **READXM** are used to write/read the complete contents of the X-Memory to/from a disk drive over HPIL. The file name must be in ALPHA. These functions exercise the full capability of the system, and provide a nice permanent backup 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.

- **PC<>RTN** is a program-pointer manipulation function. Use it to exchange it with the (last) subroutine return address. To be used with a solid understanding of their capabilities (and possible consequences).

- **VRG** reads and decodes in ALPHA the contents of the register which absolute address is in X (in program mode) or given at the prompt (in RUN mode). No stack drop is performed. Register address is checked for existence. **VRG** can be thought as the combination of **PEEKR** and **DCD** together in the same function.

- 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.

41CL 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 23) 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).

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.

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	

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

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

Stating the obvious, **MMCOPY** 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.

7.2. Buffer Utilities

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

B?	**Buffer Existence**	Buffer id# in X	W&W GmbH
CLB	**Deletes Buffer**	Buffer id# in X	W&W GmbH
BFCAT	**Buffer Catalog**	Enumerates all buffers	Angel Martin
GETBF	**Gets Buffer from file**	FileName in Alpha, id in X	Håkan Thörngren
SAVEBF	**Save Buffer to File**	FileName in Alpha, id in X	Håkan Thörngren

Not described 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, and the LastFunction id# in nibbles 5-3. 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 initial address location in the [S&X] field (nybbles 2-0) but this is of relative use at best, 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:-

When the function operates on a given buffer its id# is expected to be in the X register. This is the case for **B?** and **CLB**. The X-MEM Save/Get functions **SAVEBF** and **GETBF** also expect the File Name in Alpha.

B? will check for the existence of a buffer with id# in the X register. When executed in RUN mode the result will be YES/NO shown in the display – and if run in a program it'll follow the "do if true" rule, skipping the next program line if the buffer is not present in the system.

CLB 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.

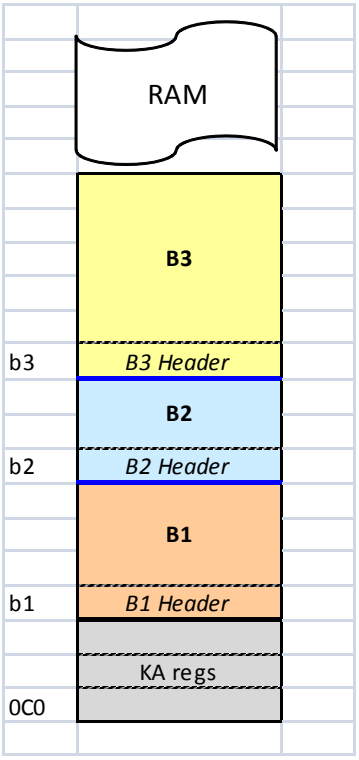
SAVEBF and **GETBF** are used for saving and Getting buffers in/from Extended memory. They follow the same convention used for other file types, with the buffer id# in X and the FileName in Alpha. Error handling includes checking for duplicate buffer ("DUP BF"), buffer existence ("NO BUF"), as well as previous File existence ("DUP FL").

Note that while it is possible to have multiple files with different (or the same) contents of one specific buffer id#, only one buffer id# can exists in the I/O RAM area at a time.

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

These functions are new versions of those in the CCD Module - using routines in the CX OS code and a novel design with the same functionality but shorter than the original ones , and possibly a tad faster in execution.

Many other functions are available in the RAMPAGE and POWERCL modules for buffer management - you're encouraged to check those as well if you want to create, resize, retype buffers, or simply view buffer contents. The list also includes Data registers and stack exchange with buffers, and much more!



7.3. 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
LKAX	**Turns Local KA On/Off**	Zero/Non-Zero in X	Gordon Pegue
MRGKA	**Merge KA to File**	FileName in Alpha	Håkan Thörngren
SAVEKA	**Save KA to File**	FileNAme in Alpha	Håkan Thörngren

The diagram below shows that each KA register can hold up to two key assignments, structured as two nybbles for the key code and four for the function id#. It also shows that they always have 0xF0 in nybbles 13 and 12 – which explains why the value 15 is not available as buffer id#.

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

- **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 OS/X calling it. You're encouraged to refer to the MLROM documentation for further details. Note that **ASG** turns on the ALPHA mode automatically upon execution, so there's no need to press it twice – this is an improvement over the standard HP-41 OS behavior.

- 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 handing 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.

  These functions are new versions of those in the CCD Module - using routines in the CX OS code and a novel design with the same functionality but shorter and possibly faster execution.

- **LKAX** is meant to be used in two ways, to temporarily suspend first and later activate the local key assignments (on keys A-J) so that they don't interfere with local program labels used in FOCAL programs. In program mode the action to perform is determined by the value in the X-register: zero will suspend the local KA, whereas any non-zero value will re-activate them. These functions only modify the key mappings in status registers 10("|-") and 15("e"), not altering the actual KA registers.

  In manual (RUN) mode the prompt will accept values 0 to de-activate them and 1 to activate them – any other will be ignored, persistently showing the prompt until a valid entry is made.

7.4. Program Branching and Flag Handling Utilities.

-AMC"OS/X	**Keycode of pressed key**	Prompts for Key	W&W GmbH
F/E	**Sets the Fix/Eng Mode**	Hybrid mode set	W&W GmbH
TF _ _	**Toggle Flag**	Allows ANY flag#	Ken Emery
TAS	**Toggles Autostart**	Toggles status	Angel Martin
GTADR _ _ _	**Goes To ROM address**	Address in Prompt / Xreg	W&W GmbH
PMTK _	**Prompts from Menu**	Choices in ALPHA	W&W GmbH
Y/N?	**Prompts for Yes/No**	Input Y/N	PANAME ROM

Menu Prompting and Program Branching.

- The function **PMTK** makes it possible to use a menu function for the HP-41. The ALPHA register is displayed and program execution is interrupted. Now the calculator is waiting for the user to press a key, and for this there are four different possibilities:

  1. The ON key turns off the calculator. The program pointer is still set to the same line, so that when starting the program again the function is executed again.

  2. A wrong key is pressed. The calculator answers this with a short sound (only when flag 26 is set).

  3. A correct key is pressed. Correct meaning that its ALPHA character is in the display. Additionally, extra text can be placed in the ALPHA register without influence on the menu control. This text must be placed in ALPHA followed by at least one blank space, and then the characters for "correct" choices. **PMTK** will distinguish between the initial informative text and the choice characters by looking at the space character separating both groups from each other. If a key whose ALPHA character is displayed is pressed, the digit value of the character is written into the X register (the leftmost character taking the value 1, and so sequentially). The stack is lifted, and this number, being position dependent, can now be used for program branching. The ALPHA register is erased, except for the commentary text and one blank space.

  4. If the ALPHA register is empty, the message "KEY?" is displayed and the key code for the next key pressed will be entered into the X register. The stack is lifted.

- The header function **–AMC"OS/X** (with XROM id# 05,00) can be used to get the keycode of a pressed key. This code is used in **PMTK** as a subroutine (see step#4 above) but it can also be used independently. The returned code will be shown in the display while the key is being depressed, and left in the X register when the key is released. If you hold the key down until the action is NULLed, the "KEY?" prompt will remain in the display, available for repeated execution; a nice way to see multiple key codes with a single initial execution of the function.

- **Y/N?** is a poor-man version that only accepts Yes/No for an answer. Back arrow cancels the function, while any other key (including ON) will simply be ignored. Answering "N" will skip the next program line, which would be used to place a GTO statement to branch the execution.

- **GTADR** sets the program pointer to the specified address: either as a two-character string in ALPHA if run in a program (i.e. "@A" = 4041 hex), or directly in the prompt (in hexadecimal). Note that the prompt length will always have four fields – irrespective of the current word size, which is therefore, unrelated to this function. This function was available in the original CCD Module under the section header "**-XF/M FNS**"

Standard and Dedicated Flag handling.

This is probably not a bad moment for a quick **flag recap**, see the table below:

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

- **F/E** sets the so called 'Fix/Eng" hybrid display mode. Flag 40 and 41 are both set. The calculator now displays all numbers as in the FIX format, but if it is so large that it needs t be expressed with exponents (greater than 9,999,999,999), it is displayed in the ENG format instead of the default SCI. Note that **F/E** does not change the number of decimal places currently configured.

- **TF** is a toggle function, inverting the status of the flag which number is in X – in a program – or entered in the prompt. It's equivalent to **IF**, the Invert Flag routine in the PPC ROM. See the PPC ROM manual pages 217 and 218 for a few useful and fun examples altering the status of the system reserved flags.

Not to be confused with the standard HP-41 flags (all of them held in status register d(14) -, the OS/X uses the scratch digits in the c(13) register (nibbles 10 and 9, between the ΣREG location and the cold start constant - see figure in next page) to host dedicated flags. The table below shows the intended use for them:

Nibble	Bit	Set	Clear
c(9)	c<18,19>	Unused / Undocumented	
	c<17>	Enabled Autostart	Disabled Autostart
	c<16>	Disabled Lower-Case mode	Enabled LC mode
c(10)	c<15>	1st. part of Prompt sequence	2nd. Part of prompt sequence
	c<14>	Unused / Undocumented	
	c<12,13>	UNS, 1CMP, 2CMP modes	

Incidentally, this usage of the scratch nibbles in register c(13) is the reason why the CCD Module (and now the OS/X Module) conflicted with other modules that also used them for other purposes, such as the ZENROM or the AECROM. I always wondered why the original programmers didn't use some digits in Buffer#5 instead for the same purposes, despite the obvious additional code complexity and relative speed penalty.

Σ	R	G	Scratch		1	6	9	R	0	0	E	N	D
13	12	11	**10**	**9**	8	7	6	5	4	3	2	1	0

Two functions are available in the OS/X Module to manage some special flags:

- **TAS** toggles the Auto-start functionality, which is the same to say the status of bit 17 in the status register c(13). This is similar to user flag 11, but the status is not changed with the first-time execution as it occurs in that case.

- **TGLC** toggles the Lower-case mode for direct character input – as described in the first section of this manual.

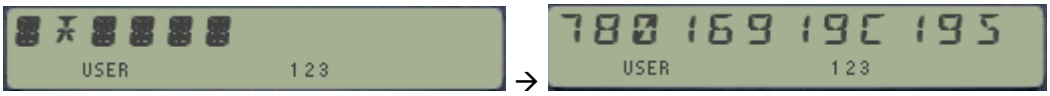
Note on the Lower-Case characters in ALPHA. These are not to be confused with the extended set of characters available to the LCD display of Half-Nut machines. Contrary to those, the Alpha characters will not be legible in program steps, or using AVIEW or when switching ALPHA on. A few functions in the ALPHA_ROM may come handy to display the ALPHA characters in the LCD display as "proper" lower-case characters, refer to that module's QRG or the PowerCL Manual for more details.

7.5. Other Miscellaneous Utilities.

The following group deals with other important functions, indispensable in every system.-

CDE	**HEX string to NNN**	Hex string in Alpha	Ken Emery
DCD	**NNN to HEX string**	NNN in X	W&W GmbH
HEXIN _	**HEX Input**	1-9, and A-F	Håkan Thörngren
MNF	**Mainframe Launcher**	Fnc. id@ in prompt	Clifford Stern
PLNG _	**Program Length**	Calculates Program size	W&W GmbH
TGPRV _	**Toggle PRIVATE status**	Program Name in Alpha	Sebastian Toelg
COMPILE	**Compiles jump distances**	Global LBL in ALPHA	Frits Ferwerda
XQ>XR	**XEQ to XROM**	Converts instructions	W&W GmbH

- Functions **CDE** and **DCD** are the ubiquitous NNN<>HEX functions present in every ROM worth its name (ML ROM, HEPAX, TOOLBOX…). We can't have enough of a good thing, or so it seems… As an example to impress your friends: decode the contents of the Status "c" register.



- **HEXIN** is another version of the well-known **HEXNTRY** function published in Ken Emery's book "MCODE for beginners", and originally written by Clifford Stern. Pretty much identical to it except that it uses the text in Alpha (if any) as prompt (useful in programs). For all purposes it supersedes **CDE**, as there's no need to first type the digits in ALPHA manually.

  The prompt will only accept hex characters, A-F and 0-9. Use Back-Arrow to delete digits and R/S to terminate the data entry. Upon termination, the corresponding NNN is placed in the X-register. HEXIN was written by Håkan Thörngren, and published in PPCJ V13N4 p13

  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.

- **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 HEPAX RAM (seen as ROM by the calculator). If Alpha is empty the program pointer must set to any line within the program. **TGPRV** is also programmable.

- **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. As an enhancement over the standard OS, the ALPHA mode will be switched on automatically for you when the function is executed.

- **MNF** is a trick of a function launcher – for mainframe functions, or should we say some of them at least. The neat thing about it is that being programmable, this becomes an unsupported and unusual way to add some of the non-programmable functions to FOCAL programs – even if the behavior is not exactly as expected.

  The table below shows some useful function id#'s to use with **MNF**. Note how in some cases this provides a way to insert in a program functions that are not programmable – circumventing the OS limitations.

MNFR #	FNCT	MNFR #	FNCT
0	**CAT**	15	**ASN**
2	**DEL**	17	$T+N<IAg _ _ _
3	**COPY**	18	e _ _
4	**CLP**	24	mie2 _ _
6	**SIZE**	27	Hb _ _
10	**PACKING**	28	mie2 _ _
12	**ALPHA**	666	**ASTO**
13	mie2 _ _	444	mie2 _ _
14	**SHIFT**	222	mie2 _ _

- **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 expedites the execution of the program (no need to search for the label), and also guarantees that short-form GOTO's are not used inappropriately.

  - There 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:

- **XQ>XR** is without a doubt also 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 to 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.

8. PROGRAM GENERATOR

This section of the manual is taken from the AECROM Manual, and describes the function **PROG**.

The OS/X Program Generator will translate your algebraic formulas into HP-41 programs. The programs that it produces are normal programs that show up in your CAT 1 listing. You can write them to cards, tape. Extended memory, or any other mass storage media; and you can use these programs as subroutines m other programs. They are in every way, sense, and form a normal HP-41 program.

The only thing unique about the programs that PROG produces is that they are created by the OS/X Module from algebraic formulas that YOU supply. With **PROG**, writing a program to solve an algebraic formula is just a matter of keying in that formula, just like it's written! You don't even have to put your HP-41 into program mode. The PROG function writes the program and stores it in memory for you to use.

8.1. A quick example of PROG

Example: As a simple first example, use the PROG function to write a program called "FRUIT" to solve the formula: FRUIT = APPLES + ORANGES

Solution:	**Keystrokes**	**Display**
	XEQ "PROG"	PROG _ (ALPHA is turned ON automatically)
	F,R,U,I,T, ALPHA	PACKING
		ENTER:FORMULA
	A, [SHIFT], +, [SHIFT], O	A+O
	R/S	ENTER:LBL,CONS.
		A. .=
	A,P,P,L,E,S	A. .=APPLES
	R/S	O. .=
	O,R,A,N,G,E,S	O. .=ORANGES
	R/S	ANS. .=
	F,R,U,I,T	ANS. .=FRUIT
	R/S	PROGRAMMING..
		PACKING
		RUN SIZE>=02

If you get the message "TRY AGAIN", you don't have enough RAM memory space for this small program. You need to either set your SIZE lower, or clear out one or more of the programs you have in your CAT 1 listing.

Running the program: Say that for this problem you have 5 apples and 6 oranges and you want to know the total amount of fruit you have. When "APPLES=?" comes up in the display, press 5, R/S. For "ORANGES=?" press 6, R/S. The calculator will temporarily display "FRUIT=" and then show you the answer: 11,000

8.2. A general description

The above formula was simple, to say the lest, but the procedure for using the **PROG** function will be no different when you use it for translating more complicated formulas into programs. The four steps for creating programs using the **PROG** function are as follows:

1. Execute PROG and, at the prompt "PROG _", supply the name for the program that you wish to appear in CAT 1. Note that ALPHA is turned ON automatically for you.

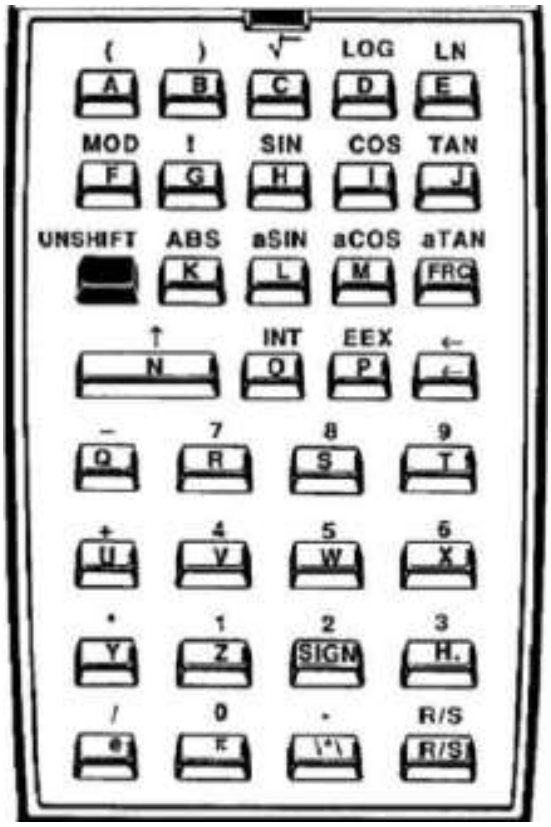
2. Key in the formula correctly (only one side of an equation) using single letters to represent variable names. (Keying in formulas is explained in greater detail below). Press R/S when you're finished.

3. Name the variables and assign values to constants. Press [R/S] after each completed input.

4. If you want the answer to be labeled. key in a name. Press [R/S], and the OS/X Module writes the program.

The key step in the above four steps is number 2. You have to know a few things about how to correctly key in a formula What functions are available and how do you key in functions? For example, how do you key in SIN(A)? Well, here are the details of keying in a formula:

When the display shows "ENTER:FORMULA", the keyboard on the HP-41 has been redefined as follows:



At first glance this keyboard appears very similar to the ALPHA keyboard. The letters are all each assigned to a key. The digits and arithmetic signs are available as shifted versions of the keys on which they're printed. But this keyboard is different from the ALPHA keyboard!

The best way to learn this new keyboard is to work with it. Execute PROG and, at the prompt "PROG _" type: T,E,S,T, [ALPHA] or any other name that you choose. The display will show: "ENTER:FORMULA".

Press the [W] key. A "W" comes up in the display. Now clear that away by pressing the back arrow key. Now press [SHIFT][W]. A "5" comes up in the display.

Press the [W] key again. Another "5" comes into the display, Notice that the SHIFT in the display hasn't cancelled. If you want the SHIFT to cancel, you have to press the shift key. This is different from the standard ALPHA keyboard, but it allows you to key in numbers like 5.775 without pressing the [SHIFT] key 5 times.

Press back arrow twice to clear those fives away, then with SHIFT on in the display press the [**J**] key. The shifted J brings the TAN function into your formula.

Continue typing to complete the formula **TAN(3A) + 0.75B + C.** As you make mistakes (say what?), you can clear them away using the back arrow key. Refer to the keyboard illustrated above to locate the characters for the above formula. Remember to press the shift key when necessary.

Switching back and forth from the shifted to the un-shifted keyboard may seem a bit awkward at first, but for keying in formulas you'll find this design to be very efficient. Once you get the above formula keyed in correctly, press the back arrow key repeatedly until you cancel the function completely.

This shows you that if you respond to the enter formula prompt by pressing the back arrow you exit the PROG function. Do not however, press R/S at that prompt - or the calculator will crash (same behavior as with the original AECROM, in case you wonder).

8.3. Keying in formulas.

Here are a few things you should notice when you are working through the following example.

1. PROG accept implicit multiplication. That is, when you key in ABC it assumes you mean A x B x C. this feature reduces the keystrokes required to key in most formulas.

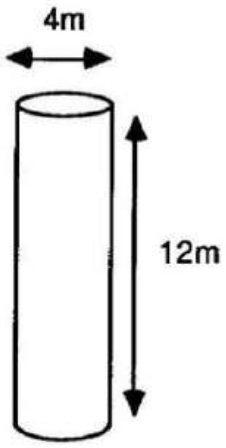
2. After you finish keying in a valid formula, **PROG** will prompt you with "ENTER:LBL,CONS"; which means you need to name your variables and assign values to any constants. At this point, the keyboard is the same as above, except that any non-character keys (like TAN, SIN, LOG) will be ignored.

3. Up to eight characters can be used to name a variable.

Example: Calculate the volume of a cylinder 4 meters in diameter by 12 meters height.

Create a program that takes the height and diameter of a cylinder and returns its volume. Don't label the answer, but name the program CYLVOL. Make use of the formula for the volume of a cylinder of a known inner-diameter and height: VOLUME = HEIGHT(π DIAM^2 /4)

Solution:	**Keytrokes**	**Display**
	XEQ "PROG"	PROG _
	C,Y,L,V,O,L, ALPHA	PACKING
		ENTER:FORMULA
	[H], SHIFT, [( ], SHIFT, [D], [^],	
	2, SHIFT, [π], SHIFT, [/], 4, [ )]	H(D^2 π /4)
	R/S	ENTER:LBL,CONS
		D. .=
	D,I,A,M	D. .=DIAM
	R/S	H. .=
	H,E,I,G,H,T	H. .=HEIGHT
	R/S	ANS. .=
	R/S	PROGRAMMING..
		PACKING
		RUN SIZE>=03



By pressing R/S when the prompt "ANS. .=" comes up in the display, you are telling **PROG** not to label the answer. The "RUN SIZE>=" prompt tells you how many registers are required to run the program. In this case you have to have at least three data registers available when you run this program.

Applying now the numeric values for this example:

XEQ "CYLVOL" , 4, R/S, 12 R/S -> 150.7964 in FIX 4.

Below is the program listing as created by PROG. Note the usage of the power function for the square power, more general than X^2. Each variable is internally associated with a data register which will be used in the calculations (so not based in the stack).

Note also that the final output doesn't combine the name of the answer with its value in the display – granted there's some finesse missing but the compromise is largely appropriate, and the methodology used quite impressive to say the least.

```
01  LBL "CYLVOL      07  STO 01       13  4
02  "DIAM=?"        08  RCL 00       14  /
03  PROMPT          09  2            15  STO 02
04  STO 00          10  Y^X          16  RCL 01
05  "HEIGHT=?"      11  PI           17  RCL 02
06  PROMPT          12  *            18  *
```

As the answer was left unnamed, the program doesn't include any steps to announce the final output. This would have been located at the end had a name been given to it at the "ANS. .=" step in data entry.

Low priority multiply [/*/]

There is a function located on the radix key that looks like this: /*/. This function is called "low priority multiply". It does the same thing as the multiply function [*], but it is evaluated after the [+] and [-] signs in your formula.

The purpose of low priority multiply is to reduce the number of parentheses in a formula that you key in. it can also save you from having to start all over when you get to the end of keying in a formula and realize that the whole expression needs to be multiplied by some value that otherwise would require the formula to be enclosed in parentheses. The example in the following section shows the use of the low priority multiply function.

Trigonometric and Hyperbolic functions.

The direct and inverse trigonometry functions are easy to locate on the keyboard and are just as easy to use. You simply key them in as you would write them in your formula on paper. Only one keystroke is necessary to key in a trigonometric function.

But, where are the hyperbolics? Yes, these functions (HSIN, HCOS, HTAN and their inverses) were included in the list of SandMath functions. You should be able to use them in your formulas, right? Certainly. Notice that the un-shifted version of the [3] key is the function [**H.**]- This key is used as a prefix to turn a trigonometric function into a hyperbolic.

Example.- Execute **PROG**, name it "TEST", and key in the formula: "$x * (\sinh^2 a + \cosh^2 b)$"

Solution: when the display reads 'ENTER:FORMULA", key in one of the following sets of keystrokes, either will work. Notice how the low priority multiply function /*/ reduces the number of parentheses in the resulting formula.

Set-1: [SHIFT], [( ], [SIN], [SHIFT], [H.], [A], [SHIFT], [ )], [^], [2], [+], [( ], [COS], [SHIFT], [H.], [B], [SHIFT], [ )], [^], 2, [/*/], [X] ; 5 times SHIFT, 22 keystrokes in total

Resulting formula: (SINH A)^2 + (COSH B)^2 /*/ X

Set-2: [SHIFT], [( ], [( ], [SIN], [SHIFT], [H.], [A], [SHIFT], [ )], [^], [2], [+], [( ], [COS], [SHIFT], [H.], [B], [SHIFT], [( ], [^], 2, [ )], [SHIFT], [X] ; 6 times SHIFT, 24 keystrokes in total

Resulting formula: ((SINH A)^2 + (COSH B)^2) * X
Finally, either press R/s to have **PROG** complete the program, or press the back arrow key repeatedly until the function cancels. Obviously the SandMath needs to be plugged in to execute it properly.

8.4. Details of PROG

PROG is non-programmable. When you key is a formula at the prompt, **PRG** insists that you follow certain rules. These rules are listed below.

1. The first character in your formula cannot be a right parentheses, MOD, FACT (!), +, *, or /. A minus sign can be used as the unary minus (for example negative 5 can be entered as –5).

2. Constants may be entered either as digits in your formula or during the 'ENTER:LBL,CONS" routine. If you wish to enter constants during the "ENTER:LBL,CONS" routine you need to include them as single letters in the formula.

3. Several functions can be followed by anything except: { +, -, *, /, FACT(!), MOD, and ")" }. Those functions are: { "(", SQRT, LOG, LN, MOD, FRC, INT, EEX, ^, +, -, *, /, SIGN, and trigonometric and HYP functions}.

4. These functions (characters in your formula) can be followed by anything: the letters A through Z, e, π, low priority multiply /*/, right parentheses, factorial (!), decimal point, and the digits 1 through 9.

When "ENTER:LBL,CONS" is displayed, up to eight characters can be keyed in to name a variable or to specify the value of a constant. You can choose to leave the single letter as the prompt for the variable (by just pressing R/s), key in a name for that variable, or key in a numeric constant.

If the first character in the name is a number or a plus or minus sign, **PROG** will take your input as a numeric constant. In a numeric constant the character "**E**" is used to signify the exponent in scientific notation (1.2E6 means 1,2 x 10^6 or 1,200,000). Also, both a comma and a dot are accepted as the radix (1.2 is the same as 1,2, which is also the same as 1,2000 so don't use commas for grouping).

When you execute **PROG** and get the prompt 'PROG _" , you have to key in a name for the program that is going to be created. You can key in any ALPHA name up to seven characters long. **PROG** always uses this name as the global ALPHA label at the first line of the program.

If you use the single characters "A" through "J" or "a" through "e", which are commonly used as local ALPHA labels, you will find that PROG still makes them into global ALPHA labels. They show up in CAT 1, but because the HP-41 expects these single letters to be local ALPHA labels, you can't access them using GTO or XEQ except, perhaps, in a synthetic program line (if you're into that sort of thing). In short, don't use those single letters as program names with **PROG** unless you enjoy the additional hassle.

Clearing programs,

The programs created by PROG can be cleared by the same methods that you use to clear any program. The HP-41 function CLP and the extended function PCLPS are dynamite when it comes to clearing programs.

Excerpt taken from the AECROM Brochure:- don't we all love the marketing department? ☺

> ***"Writes its own programs.-*** *Artificial intelligence? Close to it! The AECROM, in conjunction with the HP-41, creates its own programs to solve user-supplied equations… and fast! Simply key in the desired program name and your equation. the AECROM will automatically write the program for you. Efficient, user friendly, error free programs are written for virtually any size equation in seconds. Any number and combination of most HP-41 match functions, in addition to new hyperbolic functions, may be used in your equations".*

That's all folks, this concludes the AMC_OS/X manual. Hope you find it useful, or at least interesting to have all these functions documented at last – from the historian of the archaeological SW department to the global community with my best wishes.

Appendix 1.- Function Repeats.

The table on the right shows all functions in the OS/X module, indicating in which other modules they're also available.

The table does not include the Power_CL module, which pretty much has them all included.

The stats are as follows:

38 unique functions,
13 dup fns in the TOOLBox,
13 dup fns in the RAMPAGE.

Function	-AMC"OS/X	-TOOLBOX	Rampage	Class
ABSP	√			UTILS
ARCLH	√			UTILS
ARCLI	√			UTILS
ASG	√			KA/BUF
ASWAP	√			UTILS
B?	√			KA/BUF
BFCAT	√		√	KA/BUF
CHKSYS	√	√		UTILS
CHKROM _ _	√	√		MCODE
CODE	√			MCODE
CLA-	√			UTILS
CLB	√			KA/BUF
CLEM	√			XMEM
CMP _	√			UTILS
COMPILE	√			MCODE
DCD	√			MCODE
DCODE _ _ _	√	√		MCODE
DTOA	√			UTILS
DTST	√			UTILS
F/E	√			UTILS
GETBF	√		√	XMEM
GETKA	√		√	XMEM
GTADR	√			UTILS
HEXIN _	√	√		MCODE
MNFR _ _ _	√			UTILS
MRGKA	√		√	XMEM
MSGE	√			UTILS
OSREV	√	√		MCODE
PC<>RTN	√			UTILS
PEEKR	√		√	RAM
PG?	√	√		MCODE
PGCAT	√	√		MCODE
PGSIG _ _	√	√		MCODE
PLNG _	√			UTILS
POKER	√		√	RAM
PMTA _	√			UTILS
PMTH _ _	√			UTILS
PMTK _	√			UTILS
PROG	√			LAUNCH
RAMED	√		√	RAM
READPG	√			HPIL
READXM	√		√	HPIL
RENMFL	√		√	XMEM
RETPFL	√		√	XMEM
RNDM	√			UTILS
ROM? _ _	√	√		MCODE
ROMLST	√	√		UTILS
SAVEBF	√		√	XMEM
SAVEKA	√		√	XMEM
SEED	√			UTILS
SUMPG _	√	√		MCODE
TAS	√			UTILS
TF _ _	√			UTILS
TGPRV _	√	√		UTILS
TGLC	√			UTILS
VIEWH	√			UTILS
WRTPG	√			HPIL
WRTXM	√		√	HPIL
WSIZE _ _	√			UTILS
WSIZE?	√			UTILS
XQ>XR	√	√		MCODE
XTOAH	√			UTILS
Y/N?	√			UTILS

Appendix 2.- X-Memory File Headers.

Generally speaking, all X-Mem files have a NAME register and a HEADER register. The Name register obviously holds the file name, which is used as parameter in ALPHA for diverse file functions. The Header register is a control and status register that holds key information relevant to the file type & size, address in memory, and other auxiliary parameters – like the pointers for some file types.

The following figures show the header layout for the different file types.- Note how the file type and size (in registers) fields are common to all of them, and that those are the only fields for the "simpler" files (like Buffer, Kay Assignments, STATUS and Complex-Stack).

1. PROGRAM Files:

T	-	-	-	-	-	-	-	B	Y	T	S	Z	E
13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. DATA Files:

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

3. ASCII Files:

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

4. MATRIX Files:

T	A	D	R	L/U	C	O	L	i	j	#	S	Z	E
13	12	11	10	9	8	7	6	5	4	3	2	1	0

5. Buffer, Key-Assignment, Status-Regs, and Complex-Stack Files:

T	-	-	-	-	-	-	-	-	-	-	S	Z	E
13	12	11	10	9	8	7	6	5	4	3	2	1	0

For Data and ASCII files, the address field is initially blank – and only filled in when the pointer is set, either manually using SEEKPT(A) or automatically using some dedicated function (like GETRGX, or APPREC/CHR).

To the author's knowledge the PROGRAM Files never get the address field filled in.

Appendix 3.- X-Memory Structure.

Extended memory is comprised of up to three disjoint memory 'blocks", depending on whether only the X-Mem/Funct. module is present, or if other Extended Memory modules are also plugged into the calculator.

Each of these blocks has a "linking" registers at the bottom, holding the pointers to the previous and next block, as well as its own starting location. They are located at the bottom of each block, that is addresses 0x040, 0x201, and 0x301.

The structure of the information contained in the linking registers is shown in the figure below:

-	-	C	U	R	P	R	V	N	X	T	T	O	P
13	12	11	10	9	8	7	6	5	4	3	2	1	0

CUR: number of files; only used in bottom linking register at 0x040
PRV: address of linking register of PREVIOUS module (or zero if first block)
NXT: address top register of NEXT module (or zero if last block)
TOP: address of top register within this module

The contents of the linking registers vary depending on the number of X-Mem modules present and where they are plugged, so for instance for a full configuration (or the HP-41 CX) including 5 files in total they are as follows:

@ 0x301:

					2	0	1	0	0	0	3	E	F
13	12	11	10	9	8	7	6	5	4	3	2	1	0

@ 0x201:

					0	4	0	3	E	F	2	E	F
13	12	11	10	9	8	7	6	5	4	3	2	1	0

@ 0x040

		0	0	5	0	0	0	2	E	F	0	B	F
13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: Some of the boundary values appear to be hard-coded in the file management routines, like EMDIR, EMROOM, and file search utilities. This makes it impossible to add more blocks above - even if the memory is available (like is the case for the 41CL machine) – as shown below. Also it's unfortunately not possible to change their locations to *other pages* in RAM, say 1kB higher (for a second set of XM).

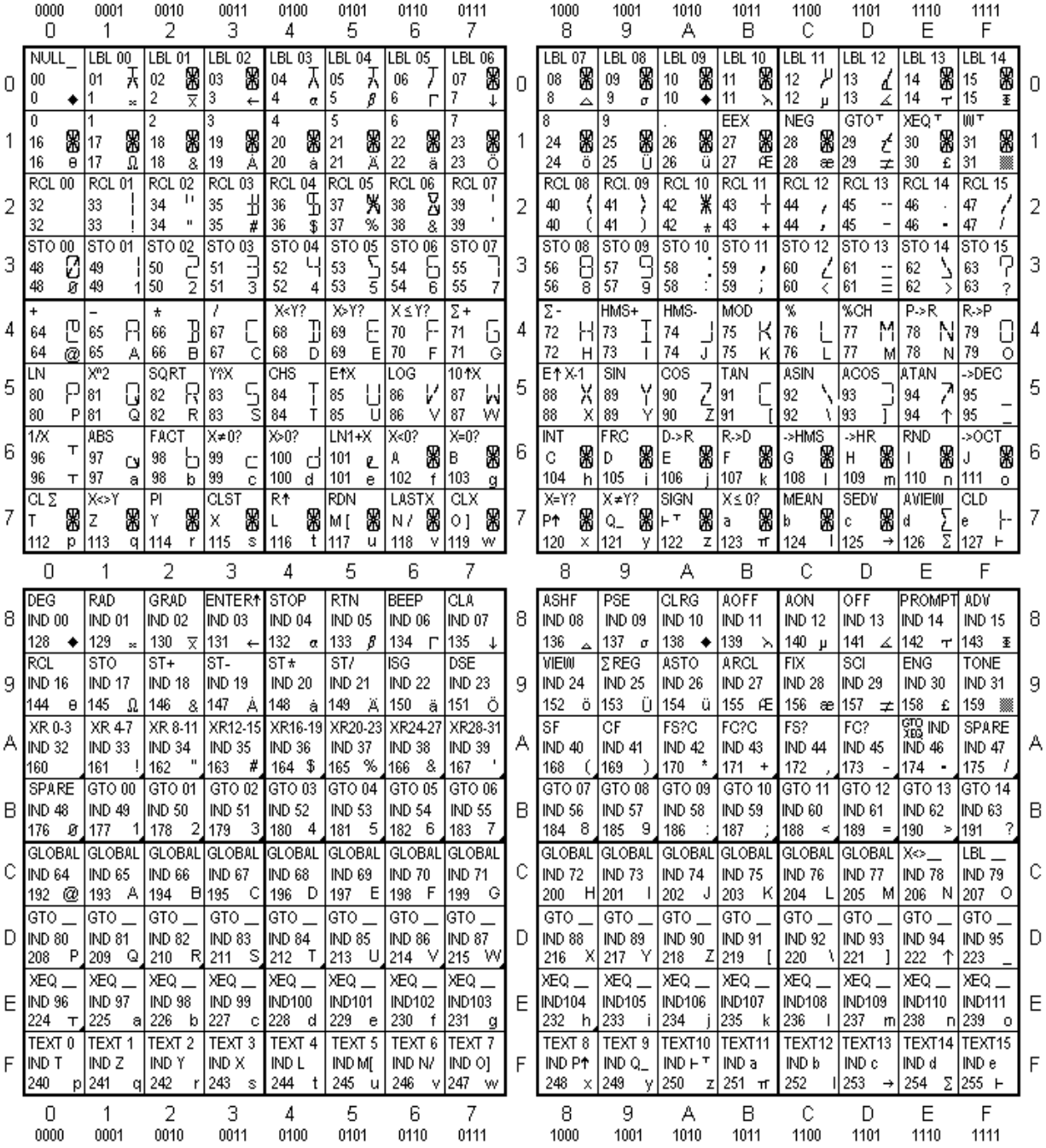
@ 0x401:

					3	0	1	0	0	0	4	E	F
13	12	11	10	9	8	7	6	5	4	3	2	1	0

@ 0x301:

					2	0	1	4	E	F	3	E	F
13	12	11	10	9	8	7	6	5	4	3	2	1	0

Apendix 4.- HP-41 Byte Table



Hex codes for bytes are the row number followed by the column.

◢ A filled lower right corner indicates a printer control character.

Bytes 90-BF, and CE-CF Prefix two-byte instructions.
Bytes D0-EF Prefix three-byte instructions.
Bytes 1D-1F, C0-CD, F0-FF Prefix variable length instructions.