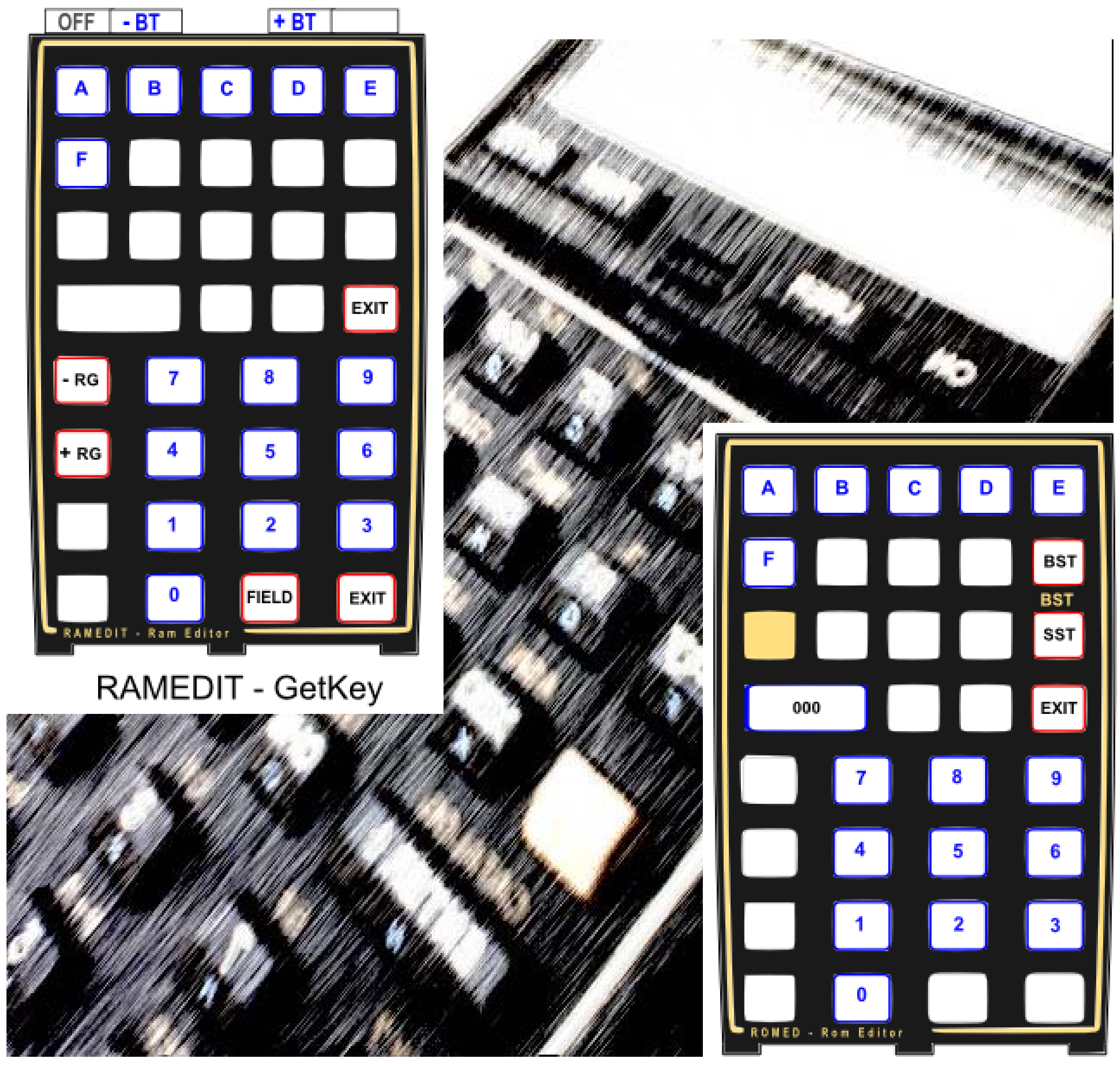
RAMPAGE & TOOLBOX Modules

Tools & Utilities for the HP-41CX

User’s Manual and QRG.



RAMEDIT - GetKey

ROMED - Wrom

Written and Programmed by Ángel M. Martin

April 2014

Acknowledgments.- This manual and the described modules would obviously not exist without the wonderful functions and routines included into them. Thanks to the MCODE pioneers and grand masters who published their work in PPC Journal and other sources, such as Ken Emery (and alter-ego Skiwd), Clifford Stern, Doug Wilder, Håkan Thörngren, Frits Ferwerda and Nelson F. Crowle amongst others for their powerful functions, real examples of solid MCODE programming.

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.

RAMPAGE & TOOLBOX Modules

Table of Contents.

4. TOOLBOX – I: System Info Utils

5. TOOOBOX – II: Hacker’s Lab

Appendices.

RAMPAGE & TOOLBOX Modules

Tools & Utilities for the HP-41CX

1. Introduction.

This manual documents two modules, the RAMPAGE and the TOOLBOX. These modules are completely independent from one another, yet you'll likely find yourself using them together or in combination – therefore the treatment as a single unit for the purposes of the documentation. Their areas of application are loosely defined by RAM-related utilities in the RAMPAGE case, and system and ROM-related tools for the TOOLBOX.

These modules are result of the logical evolution that started with the SANDBOX module, back in 2002 – so here it is all rounded up, twelve years latter. Amongst the included functions you'll find the usual suspects: powerful ROM and RAM Editors, 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 plenty of other system and X-Mem utilities.

The modules are a "greatest hits" compilation of functions from several sources. You'll recognize some from the ZENROM, and Doug Wilder's DISASM/BLDROM but mostly are MCODE jewels published on the PPC Journals. It comes without saying that only a fraction of the functions are written by the author – although I can say I've tweaked them all to take advantage of the Library#4 and general arrangements.

This manual is structured around the four sections of the modules (two sections each), as follows:

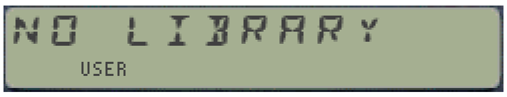
• RAMPAGE'X	Includes general-purpose RAM utils, RAM Editors and X-Men extensions.
• KA/BUF FNS	Buffer creation and management tools. Plus additional handy KA utils.
• TOOLBOX"4X	General-purpose system info and details
• HACKERS LAB	Advanced functions for programmers (FOCAL and MCODE)

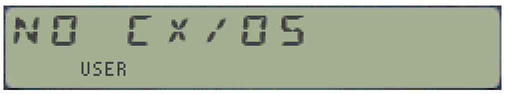
Page#4 Library (but not Bank-Switching.)

The modules both use the Library#4 – but they are simple configurations, not-bank switched. Using the Library#4 allowed for a substantial increase in the number and kind of functions compared with the initial incarnations, yet the most limiting factor always is the number of functions in the FAT (64 maximum). Moving to a bank-switched configuration would have required an auxiliary FAT with sub-functions; an approach used in other modules (POWERCL, SANDMATH, etc) that adds further complexity to the design and imposes a few restrictions to the use and applicability. I opted for a simpler design in this case, using two pages and two main FAT's. – So sorry folks, but no fancy new overlays, launchers or sub-function groups this time!

The last remark is regarding the CX dependency: they are designed for the CX version of the 41 OS, as the code profusely uses subroutines from the CX OS code. This was a compromise to maximize the functionality and the economy of ROM space – as it avoided having to replicate large code streams already available on the CX. Do not use these modules on a 41C or CV machine, it'll have unexpected and unwanted results.

The modules check for the presence of their dependencies, i.e. the Library#4 and the CX.-- if the Library#4 is missing or the machine is not a CX the errors will halt it to avoid likely problems. Note also that these modules are not compatible with page#6 – avoid plugging them in that location.


NO LIBRARY
USER


NO CX/OS
USER

Remember: The RAMPAGE and TOOLBOX modules extensively use 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 128 functions: two full-house FATs with the best tools in town. Original authors are listed (to the best of my knowledge).

RAMPAGE Module: RAM Tools and Editors, KA & Buffer Management.

Function Name	Description	Input	Output	Author
-RAMPAGE'X	*Header*	*None*	*Shows splash Lib#4 splash*	*Nelson F. Crowle*
A<>RG _ _	Swaps Alpha and Regs.	prompts for RG#	RG swapped	*Ángel Martin*
A<>ST	Swaps Alpha and Stack	None	ST swapped	*Ángel Martin*
ARCLCHR	ARCL Char	FileName in Alpha	Char appended to Alpha	*Håkan Thörngren*
ARCLIP _ _	ARCL Integer Part	decimal number in X	ARCL integer part	*Frits Ferwerda*
CLMM	Clear Main Memory	needs OK in Alpha	Main Memory Deleted	*Zengrange*
CLRAM	Clears RAM	needs OK in Alpha	All RAM Deleted	*Raymond dTondo*
CLXM	Clear Extended Memory	needs OK in Alpha	EM deleted	*Zengrange*
FLCOPY	Copy File	“Source,Destination” in Alpha	File Contents copied	*Ángel Martin*
FLHD	File Header	FileName in Alpha	Header Address in X	*Ángel Martin*
FLTYPE	File Type	FileName in Alpha	Type in X	*Ángel Martin*
GETST	Get Status Rgs	FileName in Alpha	Restores all Status registers	*Ángel Martin*
NRCLX _ _	non-Normalized RCL	Register# in prompt	contents in X	*Syd Kelly*
PEEKR	NNN Recall	Absolute address in X	NNN in X	*Ken Emery*
POKER	NNN Store	Absolute address in X	NNN in Y	*Nelson F. Crowle*
RAMED _	RAM Editor	Address in M or PRGM pointer	RAM Editor	*Zengrange*
RAMEDIT _	RAM Editor	Address in X or PRGM pointer	RAM Editor	*Håkan Thörngren*
RCLBM	Recall Byte by M	Address in M	Byte in X	*Mark Power*
READXM	Reads EM from MassStg	FileName in Alpha	All EM Restored	*Skwid*
RENMLFL	Rename File	OldName, NewName in Alpha	File renamed	*Ángel Martin*
RETPFL	Re-type File	Old, New types in X	File re-typed	*Ángel Martin*
RSTCHK	Reset Checksum	Program FileName in Alpha	Checksum Restored	*Håkan Thörngren*
ST<>Σ	Swaps Stack and SRG	None	RG swapped	*Ángel Martin*
ST<>RG _ _	Swaps Stack and RG	prompts for RG#	RG swapped	*Ángel Martin*
SAVEST	Save Status Registers	FileName in Alpha	All Status registers saved	*Ángel Martin*
STOBM	Store Byte by M	Address in M, Byte in X	Stores Byte	*Mark Power*
WORKFL	Get Work File	none	WorkFile name appended	Sebastian Toelg
WRTXM	Write EM to MassStg	FileName in Alpha	All EM written	*Skwid*
X<>aNN	NNN Exchange	Absolute address in X	NNN in Y	*Nelson F. Crowle*
X<>BM	Exchange Byte by M	Address in M, Byte in X	Exchanged values	*Mark Power*
X<I>Y	Swaps Ind(X) and Ind(Y)	pointers in Y & X	swapped values in Regs.	*Nelson F. Crowle*
XQXM	Execute XM Program File	FileName in Alpha	Program will execute	*Ross Wentworth*
-KA/BUF FNS	***Buffer Finder***	***Buffer id# in X***	***Yes-No, skip if false***	***Ángel Martin***
ARCLBF _ _	ARCL Buffer	Buf id# in prompt	Buffer content to Alpha	*Ángel Martin*
ASTOBF _ _	ASTO Buffer	Buf id# in prompt	Alpha to Buffer	*Ángel Martin*
BF<>RGX _ _	Swap Buffer and Registers	Buf id# in prompt	Swaps Buffer and Regs.	*Ángel Martin*
BF>ST _ _	Buffer to Stack	Buf id# in prompt	***Buffer contents to Stack***	*Ángel Martin*
BFCAT	Buffer Catalog	none	***Enumerates all buffers***	*Ángel Martin*
BFHEAD	Buffer Header Content	Buffer id# in X	Decodes Header Register	*Ángel Martin*
BFLNG	Buffer Length Finder	Bufferid# in X	Number of registers used	*Ángel Martin*
BFRCL _ _	Recall Buffer from RG	id# in X, prompts for RG#	Buffer restored from RG	*Ángel Martin*

Function Name	Description	Input	Output	Author
BFSTO _ _	Buffer Address/Size	id# in X, prompts for RG#	Buffer Saved in RG	*Ángel Martin*
BFVIEW _ _	Buffer View	Buf id# in prompt	Shows buffer registers	*Ángel Martin*
BLIST	Show Buffers	none	String with existing Buffers	*David Yerka*
BUFHD	Buffer Header Address	Buffer id# in X	Header Address in X	*Ángel Martin*
CLBUF	Clear Buffer by X	Buffer id# in X	Clears buffer Contents	*Ángel Martin*
CRBUF	Creates Buffer	id#,size in X	Creates Buffer	*Ángel Martin*
DELBUF	Deletes Buffer	Buffer id# in X	Deletes buffer	*Ángel Martin*
GETBF	Restores Buffer from EM	FileName in Alpha	Buffer restored	*Håkan Thörngren*
GETKA	Get Keys	FileName in Alpha	Key Assignments Restored	*Håkan Thörngren*
KACLR	Clear Key Assignments	OK or OKALL in Alpha	Deleted buffer(s)	*Hajo David*
KALNG	KA Length Finder	None	Number of registers used	*W&W GmbH*
KAPCK	Pack Key Assignments	None	Packed KA buffer	*Hajo David*
KYOFF	Key Assignment Off	Press key at prompt	KA suspended	Frits Ferwerda
LKAOFF	Suspend Local KA	None	Deactivates A-J assignments	*Ross Cooling*
LKAON	Activate local KA	None	Reactivates A-J assignments	*Ross Cooling*
MRGKA	Merge Keys	FileName in Alpha	Key Assignments Merged	*Håkan Thörngren*
REIDBF	Re-issue buffer id	Old, new id's in X	Changes id#	*Ángel Martin*
RESZBF	Resize Buffer	id#,size in X	Buffer resized	*Ángel Martin*
RGX>BF _ _	Registers to Buffer	Buf id# in prompt	Registers saved in Buffer	*Ángel Martin*
SAVEBUF	Saves Buffer in EM	id# in X, FileName in Alpha	Buffer Saved in EM	*Håkan Thörngren*
SAVEKA	Save Keys	FileName in Alpha	Key Assignments Saved	*Håkan Thörngren*
ST>BF	Stack to Buffer	Buf id# in prompt	Stack saved in Buffer	*Ángel Martin*
VKEYS	View Keys	None	Shows KA catalog	*Stephane Barizienne*

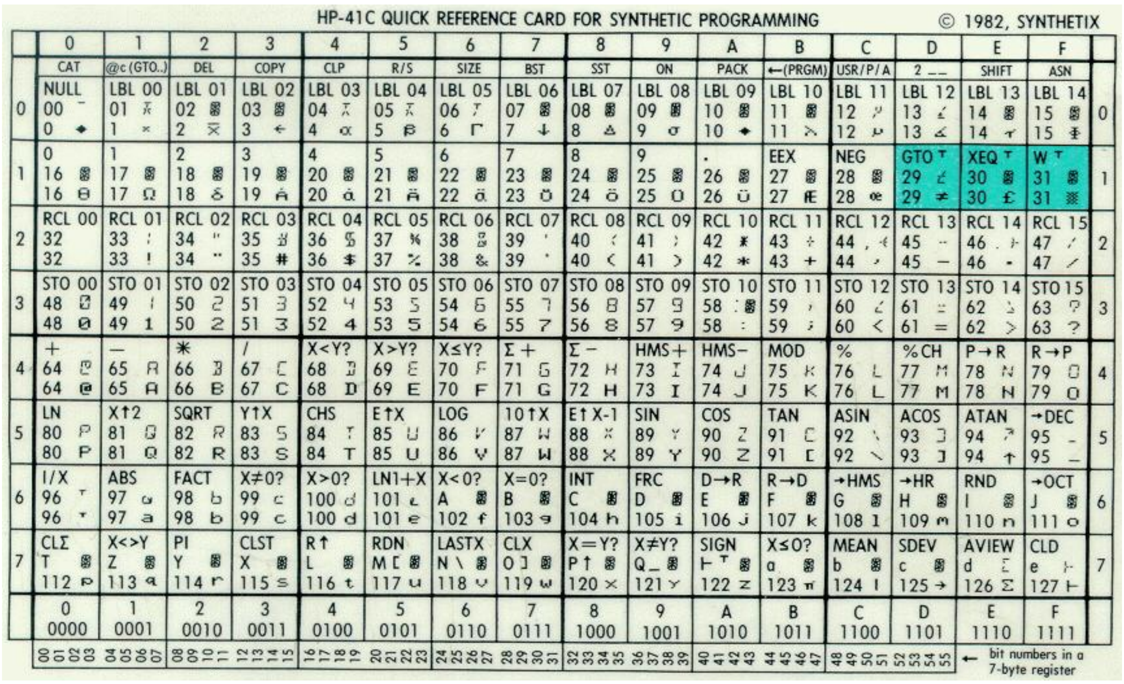
TOOLBOX Module.- General System Utils and Advanced Programming.

Function Name	Description	Input	Output	Author
-TOOLBOX 4	*Header*	*none*	*Shows Lib#4 splash*	*Nelson F. Crowle*
ΣRG?	Stat Regs Finder	None	Stat Regs Address in X	Ken Emery
APPEND	Append function	none	same as pressing ALPHA, Append	Doug Wilder
BST^	non-stop BST	none	lists prgm lines while pressed	Nelson F. Crowle
CFX _ _	Clear Flag	flag number in prompt	flag status cleared (0-55)	Michael Katz
CRTN?	Curtain Finder	None	Curtain Address in X	N/A
CSST	Continuous SST	Program Pointer positioned.	Program lines displayed	Phi Trinh
CVIEW	Continuous View	non-stop AVIEW	Shows contents of Alpha	Frits Ferwerda
DREG?	Data Registers Finder	None	Current Size	Ken Emery
DSP _	Sets decimal places	Prompts for number	settings made	Sebastian Toelg
DSP?	Recalls decimal places set	none	current decimal places in X	Ángel Martin
FC?S _ _	Is flag set / set	flag number in prompt	does the logic	Ken Emery
FS?S _ _	is flag clear / set	flag number in prompt	does the logic	Ken Emery
FREG?	Free Registers Finder	None	Available Main Memory registers	Ken Emery
FSIZE?	HPIL Media File Size	File Name in Alpha	HPIL File size in X	Eramco?
GTEND	Go to .END.	none	Goes to the permanent .END.	Ken Emery
LASTP	Last Program	none	Goes to last program in RAM	Zengrange
NOP	No Operation	none	Inserts a "F0" line in program	Zengrange
PGCAT	Page Catalog	None	Catalogs all Pages	VM Electronics
PTCAT _	Port Catalog	Port number (1,2,3,4)	Catalogs ROM functions	J.D. Dodin
READF	Read Data File	HPILName, EM Fname	Copies data from HPIL to EM	R. del Tondo
ROMCAT	ROM Catalog	ROM ID# in X	Catalogs ROM functions	J.D. Dodin
ROMLST	Shows all ROM id´s	none	List in Alpha	Ángel Martin
SFX _ _	Set Flag	flag number in prompt	Flag status set (0-55)	Michael Katz
SST^	Continuous SST	set pointer in program	Lists prgm lines while pressed	Nelson F. Crowle
TGFX _ _	Toggle Flag	flag number in prompt	Flag status toggled (0-55)	Ken Emery
TGPRV _	Toggle Private	Program name in Alpha	Program Status Changed	Sebastian Toelg
WRTDF	Write Data File	XMFName,ILFName in Alpha	Copies data file to HPIL	R. del Tondo

Function Name	Description	Input	Output	Author
XQ>GO	Pop Return Address	none	Destroys First Return Address	Håkan Thörngren
XROM _ _/_ _	Input XROM function	Promps for RR:FF	Executes the function	Clifford Stern
-HACKER LAB	Tests for subroutine RTN	none	YES/NO, skips line if False	Doug Wilder
ADR? _ _ _ _	Address encoder	4-digit address in prompt	encoded NNN in X	Doug Wilder
BCDBIN	BCD to Binary	BCD in X	NNN in X	Ken Emery
BINBCD	Binary to BCD	NNN in X	BCD in X	Ken Emery
BLANK?	Tests page for blank	pg# in X-reg	YES/NO	Ángel Martin
CGO _ _ _ _	?C GO encoder	4-digit address in prompt	result in display	Doug Wilder
CHKSYS	Check System	none	Checks ROM conflicts	Ángel Martin
CHKROM _ _	Check ROM	XROM Number	Test result message	HP Co.
CLBL	Clear Block	BBBB|EEEE in X as NNN	Block cleared. Expects "OK"	Friitz Ferwerda
COMPILE	Compiles program	Program name in Alpha	All GTO/XEQ are compiled	Frits Ferwerda
CPYPG _ _	Copy Page	source in X, dest. In prompt	Copies pg in x to pg in prompt	Ángel Martin
CXQ _ _ _ _	?C XQ Encoder	4-digit address in prompt	result in display	Doug Wilder
DCODE _ _ _	Decode	NNN in X / Prompt	Hex in Alpha	Clifford Stern
DISSST	SST Disassembler	Begin/End in prompts	disassembled output	VM Electronics
FDATA _	Function Data	Prompts for Function Name	Address, Type, asgn data in Alpha	Klaus Huppertz
FNC?	Function Stats	XROM#,FNC# in X	results in Alpha, X, and Y	W&W GmbH
GETW	Get Word	Absolute ROM address in X	Word Value in X as NNN	Ángel Martin
HEX>VSM _	HEX to VASM Oct	Prompts for Hex in Alpha	Oct in Alpha in VASM format	Ken Emery
HXENTRY _	Enter NNN Directly	Prompts for Hex in Alpha	NNN in X	Clifford Stern
HEXIN _	Enter NNN Directly	Prompts for Hex in Alpha	NNN in X	Håkan Thörngren
IOBUS	I/O Bus Information	0,1,2, in prompt	Free, Used, or banked pages	Ángel Martin
JC	JC Encoder	distance in X	result in display	Doug Wilder - Á. Martin
JNC	JNC Encoder	distance in X	result in display	Doug Wilder - Á. Martin
NCGO _ _ _ _	?NC GO Encoder	4-digit address in prompt	result in display	Doug Wilder
NCXQ _ _ _ _	?NC XQ Encoder	4-digit address in prompt	result in display	Doug Wilder
NOPS	Finds NOP´s	BBBB|EEEE as NNN in X	shows address and no. of NOPS	Frits Ferwerda
PG? _ _	Page Stats	Page number in prompt	Pg# in prompt	W&W GmbH
PGROOM	Available Room in Page	Page# in prompt	total number of bytes left	Ángel Martin
PGSIG _ _	ROM Signature	Port number (dec)	Trailing string in Alpha	Ángel Martin
ROM? _ _	ROM Stats	ROM id· in prompt	Rom id# in prompt	Frits Ferwerda
ROMED	ROM Editor	Prompts for address	Edit mode	Doug Wilder
SUMPG _ _	Checksum Calculation	Prompts for Page number	Checksum updated	George Ioannou
VSM>HEX _	VASM Oct to HEX	Prompt for Oct in Alpha	Hex in Alpha	Ken Emery
XQ>XR	XEQ to XROM	Program name in Alpha	XEQ changed to XROM	W&W GmbH

Now, take a breath and have a repeat look at the QRG's above to realize there's a lot of material to cover ahead in the manual – something to look forward to, I hope. Perhaps you're missing some of the all-time favorites functions from the CCD Module, are you? Well, that's not an overlook or negligence on my part: the CCD has spawned an entire new module by itself, the AMC_OS/X Module with the "best and the rest" of CCD-related material. You're encouraged to check that one out in the unlikely case you haven't yet and aren't using it as a permanent fixture on your 41 configurations.

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



HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING © 1982, SYNTHETIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	CAT	@c (GTO..)	DEL	COPY	CLP	R/S	SIZE	BST	SST	ON	PACK	←(PRGM)	USR/P/A	2 __	SHIFT	ASN	
0	NULL 00 ‾ 0 ◆	LBL 00 01 x̄ 1 ×	LBL 01 02 ▩ 2 x̄	LBL 02 03 ▩ 3 ←	LBL 03 04 x̄ 4 α	LBL 04 05 x̄ 5 β	LBL 05 06 Γ 6 Γ	LBL 06 07 ▩ 7 ↓	LBL 07 08 ▩ 8 Δ	LBL 08 09 ▩ 9 σ	LBL 09 10 ▩ 10 ◆	LBL 10 11 ▩ 11 λ	LBL 11 12 μ 12 μ	LBL 12 13 ∠ 13 ∡	LBL 13 14 ▩ 14 τ	LBL 14 15 ▩ 15 Φ	0
1	0 16 ▩ 16 θ	1 17 ▩ 17 Ω	2 18 ▩ 18 δ	3 19 ▩ 19 Ȧ	4 20 ▩ 20 ȧ	5 21 ▩ 21 Ä	6 22 ▩ 22 ä	7 23 ▩ 23 Ö	8 24 ▩ 24 ö	9 25 ▩ 25 Ü	. 26 ▩ 26 ü	EEX 27 ▩ 27 Æ	NEG 28 ▩ 28 œ	GTO ᵀ 29 ∠ 29 ≠	XEQ ᵀ 30 ▩ 30 £	W ᵀ 31 ▩ 31 ▒	1
2	RCL 00 32 32	RCL 01 33 ! 33 !	RCL 02 34 " 34 "	RCL 03 35 # 35 #	RCL 04 36 $ 36 $	RCL 05 37 % 37 %	RCL 06 38 & 38 &	RCL 07 39 ' 39 '	RCL 08 40 (40 (	RCL 09 41) 41)	RCL 10 42 * 42 *	RCL 11 43 + 43 +	RCL 12 44 , , 44 ,	RCL 13 45 - 45 —	RCL 14 46 . , 46 .	RCL 15 47 / 47 /	2
3	STO 00 48 0 48 0	STO 01 49 1 49 1	STO 02 50 2 50 2	STO 03 51 3 51 3	STO 04 52 4 52 4	STO 05 53 5 53 5	STO 06 54 6 54 6	STO 07 55 7 55 7	STO 08 56 8 56 8	STO 09 57 9 57 9	STO 10 58 : ▩ 58 :	STO 11 59 , 59 ;	STO 12 60 < 60 <	STO 13 61 = 61 =	STO 14 62 > 62 >	STO 15 63 ? 63 ?	3
4	+ 64 @ 64 @	— 65 A 65 A	✱ 66 B 66 B	/ 67 C 67 C	X<Y? 68 D 68 D	X>Y? 69 E 69 E	X≤Y? 70 F 70 F	Σ+ 71 G 71 G	Σ− 72 H 72 H	HMS+ 73 I 73 I	HMS− 74 J 74 J	MOD 75 K 75 K	% 76 L 76 L	%CH 77 M 77 M	P→R 78 N 78 N	R→P 79 O 79 O	4
5	LN 80 P 80 P	X↑2 81 Q 81 Q	SQRT 82 R 82 R	Y↑X 83 S 83 S	CHS 84 T 84 T	E↑X 85 U 85 U	LOG 86 V 86 V	10↑X 87 W 87 W	E↑X-1 88 X 88 X	SIN 89 Y 89 Y	COS 90 Z 90 Z	TAN 91 [91 [	ASIN 92 \ 92 \	ACOS 93] 93]	ATAN 94 ↑ 94 ↑	→DEC 95 _ 95 _	5
6	1/X 96 ᵀ 96 ᵀ	ABS 97 a 97 a	FACT 98 b 98 b	X≠0? 99 c 99 c	X>0? 100 d 100 d	LN1+X 101 e 101 e	X<0? A ▩ 102 f	X=0? B ▩ 103 g	INT C ▩ 104 h	FRC D ▩ 105 i	D→R E ▩ 106 j	R→D F ▩ 107 k	→HMS G ▩ 108 l	→HR H ▩ 109 m	RND I ▩ 110 n	→OCT J ▩ 111 o	6
7	CLΣ T ▩ 112 p	X<>Y Z ▩ 113 q	PI Y ▩ 114 r	CLST X ▩ 115 s	R↑ L ▩ 116 t	RDN M [▩ 117 u	LASTX N \ ▩ 118 v	CLX O] ▩ 119 w	X=Y? P ↑ ▩ 120 ×	X≠Y? Q _ ▩ 121 y	SIGN ⊢ ᵀ ▩ 122 z	X≤0? a ▩ 123 π	MEAN b ▩ 124 ǀ	SDEV c ▩ 125 →	AVIEW d Σ 126 Σ	CLD e ⊢ 127 ⊢	7
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	
	00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	← bit numbers in a 7-byte register		

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

2. RAMPAGE – General-purpose Utilities.

2.1.- RAM CLEARING & EXCHANGE

Let's open up the manual with an easy selection of RAM-related utilities, for register exchange and convenient block data handling.

A<>RG _ _	Swap ALPHA and Registers	Initial RG# in prompt	Ken Emery
A<>ST	Swaps Alpha and Stack	No inputs needed	Ángel Martin
ARCLIP _ _	Appends Integer part to Alpha	RG# in prompt	Ángel Martin
ST<>Σ	Stack swap with Stat Regs	Uses current SREG setting	Nelson F. Crowle
ST<>RG _ _	Stack swap with Data Regs.	Initial RG# in prompt	Angel Martin

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

- **ARCLIP** appends to ALPHA the integer part of the number in register specified at the prompt. Perfect to append indexes and counter values without having to change the display settings (FIX 0, CF 29). This is similar to functions **AINT**, **AIP**, and **ARCLI**, except that these operate on the X-register instead.

- **ST<>RG** and **ST<>Σ** are also register block exchange routines, which swap the stack with your choice of data registers (4 registers in total) or with the statistical registers respectively (five registers, including LASTX as well).

The existence of the highest-number register is always checked, resulting in the "NONEXISTENT" error message if not available. Should that occur, you need to change the SIZE settings or make more data registers as needed.

2.1.1. Using Non-Merged Functions in Programs.

Note that these prompting functions are programmable. When used in a program they take the argument from the next program line, a technique known as "non-merged" program lines. This has the obvious advantage of not using the X-register to hold the argument, which would defeat the purpose of stack-related functions. If the second line is not a number, the function assumes zero for argument.

For example, to swap the stack and data registers R00 to R03 simply use **ST<>RG** typing any numbers for the prompt, they will we ignored in PRGM mode. To swap the Alpha {M,N,O,P} registers and data registers R05 to R08 you need two program lines, as follows:

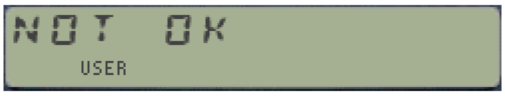
```
nn      A<>RG
nn+1    5
```

This technique was first used on the HEPAX module, but this implementation is based on Doug Wilder's routines. Be aware that the preceding line cannot be a test function (YES/NO, skip if false) for obvious reasons.

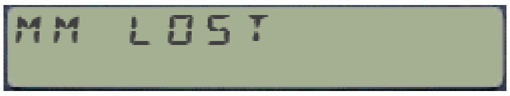
5.1.2. Clearing Memory – selectively or wholesale

CLMM	Clears Main Memory	Clears Data RGS, KA, buffers	Zengrange
CLRAM	Clears ALL RAM	Same as MEMORY LOST (!)	R. del Tondo
CLXM	Clears X-Memory	Clears all X-Mem files	Zengrange
"VREG"	View Registers	Control word bbb,eee in X	Ángel Martin

Use these functions carefully – there's no way back and what you erase cannot be recovered (no UNDO button!). To avoid unintentional uses, these functions expect the string "OK" or "OKALL" in ALPHA. If that confirmation string isn't there the execution will abort showing the error message below:

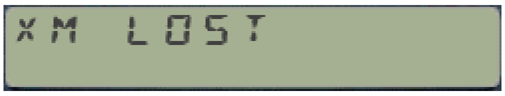

NOT OK
USER

- **CLMM** erases the calculator Main memory, including Stack & Data Registers, Programs, and I/O Area - Key assignments and buffers (Alarms included). It will however leave X-Memory untouched. Note that **CLMM** stores nulls into every register, and in addition all status registers and flags are restored to default states. The size of program memory will be 219 on an HP-41CX. Executing it from a running program will cause the program to stop – even if that program is synthetically made to run in Extended Memory, and as such is not erased) because the program pointer will be reset to point to the .END., causing the program to halt.


MM LOST

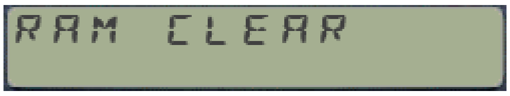
is generated when executed.

- **CLXM** erases all files in extended memory, all gone for good! Note that this function is very similar to **CLEM**, available in the AMC_OS/X and PowerCL modules – the only subtle difference is that **CLXM** overwrites the contents of all existing Extended Memory registers with nulls, whereas **CLEM** only erases the main X-Mem register but not the actual contents, so at least in theory you could restore things if you'd made a backup of the header register first (which in all practicality, nobody does of course).


XM LOST

is generated when executed in RUN mode.

- **CLRAM** wipes off the complete calculator RAM, therefore like both above functions combined together. This is similar to the MEMORY LOST condition indeed.


RAM CLEAR

is generated when executed in RUN mode

- **VREG** is a short FOCAL routine that sequentially shows the contents of the data registers specified by the control word "bbb,eee" in the X-register. A short pause is made in-between each display, you can stop it and resume it using R/S – This function is an example where MCODE would not be of any real advantage.

  Note: VREG was replaced by FLCOPY in the lasts version of the RAMPAGE module. See the FOCAL listing in the appendix-1 if you're interested.

Accessing Bytes and Registers Directly.

NRCLX	Non-Normalized Data RG# RCL	Data RG# in prompt	Sid Kelly
PEEKR	Absolute address RCL	Absolute address in X-reg	Ken Emery
POKER	Absolute address STO	Abs. adr in Y, content in X	Ángel Martin
RCLBM	Byte Recall by M	Absolute address in M	Mark Power
STOBM	Byte Storage by M	Abs. adr in M, content in X	Mark Power
X<>BM	Byte Exchange by M	Abs. adr in M, content in X	Mark Power
X<I>Y	Indirect exchange by X and Y	RG# in X and Y	Nelson F. Crowle
SWAPR	Non-normalized RG# Exchange	Data RG# in prompt	Ángel Martin

Use functions in this group to access individual registers or bytes within memory. The input arguments can be either absolute addresses or the relative data register numbers. They are related by the equation: ABS = RG# + R00#, where R00# is the absolute location of data register R00 – as returned by the function **CRTN?** (the "Curtain" locator). Absolute addresses can range from zero to 0xFFF hex (4,095 in decimal)

- **NRCLX** does a non-normalized RCL of the data register which *number* is entered in the prompt; or in the X-reg if used in a program. The stack is lifted.

- **SWAPR** does a non-normalized exchange between the Y-register and the data register which *number* is entered in the prompt; or in the X-reg if used in a program.

- **X<I>Y** is an all-indirect register exchange, using the RG# stored in X and Y. For example, to exchange registers 13 and 32 you would use the following sequence (much more convenient than using standard functions RCL IND, and X<>): 13, ENTER^, 32, **X<I>Y**

- **RCLBM**, **STOBM**, and **X<>BM** apply to byte-level handling (STO, RCL, <>) rather than to whole registers. The byte value is expected in the X-reg. They use the ALPHA register M to hold the byte absolute address as a string, according to the ZENROM convention. This consists of two alpha characters which corresponding hex codes represent the memory address (i.e. B:RRR in hex). Written by Mark Power, and published in Data File V7 N8 p24.

  The NONEXISTENT error indicates that the byte value in X is greater than 255, or that the byte position in M is greater than 6, or that the specified register does not exist.

  For example, to recall the contents of flags 0-7 (sixth byte in status register 14) and clear them, perform the following: X will contain old values (in decimal), and flags 0-7 will be clear

  HEXIN "600E", R/S, STO M, CLX, XEQ "**X<>BM**" , or alternatively
  CLA, 96, **XTOA**, 14, **XTOA** , CLX, XEQ "**X<>BM**"

- **PEEKR** can be compared to the RCL function, however it is possible to read the contents of any register without normalization into the X register. The register to be read is entered as absolute address in to X. The stack is 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 – using function **CTRN?**

- **POKER** writes over the register which absolute address is specified in the Y register, with the NNN 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 16) to addresses located 1k above them. In fact, one can have an alternate set of memory and "swap" between both as needed, duplicating so the calculator's on-line capacity. An additional benefit is that *the secondary set will not be affected in case of a MEMORY LOST*, thus you can use it as a safety backup as well.

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

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

1	LBL "MMSWAP"			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*		

And 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).

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	

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.

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.

2.2 RAM EDITORS

RAMEDIT	**RAM Editor**	Uses GETKEY [KEYFNC]	Håkan Thörngren
RAMED	**RAM Editor**	Uses [NEXT]	ZENROM

RAM editors are no doubt amongst some the best examples ever written for the HP-41 system, and as such not one but two are included in the module.

2.2.1. Editing RAM memory with RAMEDIT.

Written by MCODE master Håkan Thörngren, this powerful RAM editor is my preferred choice, as it 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.

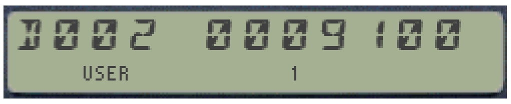
The starting register address is taken from the X register in RUN mode either as a decimal value between 0 and 999, or an a NNN with the address in the rightmost two bytes The latter form alloes for a direct entry to a byte value within the register. In PROGRAM mode it uses the current program pointer instead.

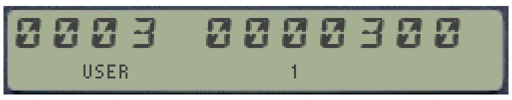
The display shows two distinct fields, with the nybble & byte section shown on the left side and the actual register content shown on the right – as a 7-digit scrollable field controlled by the USER and PRGM keys – very much like the CX’s ASCII file editor **ED**.

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.





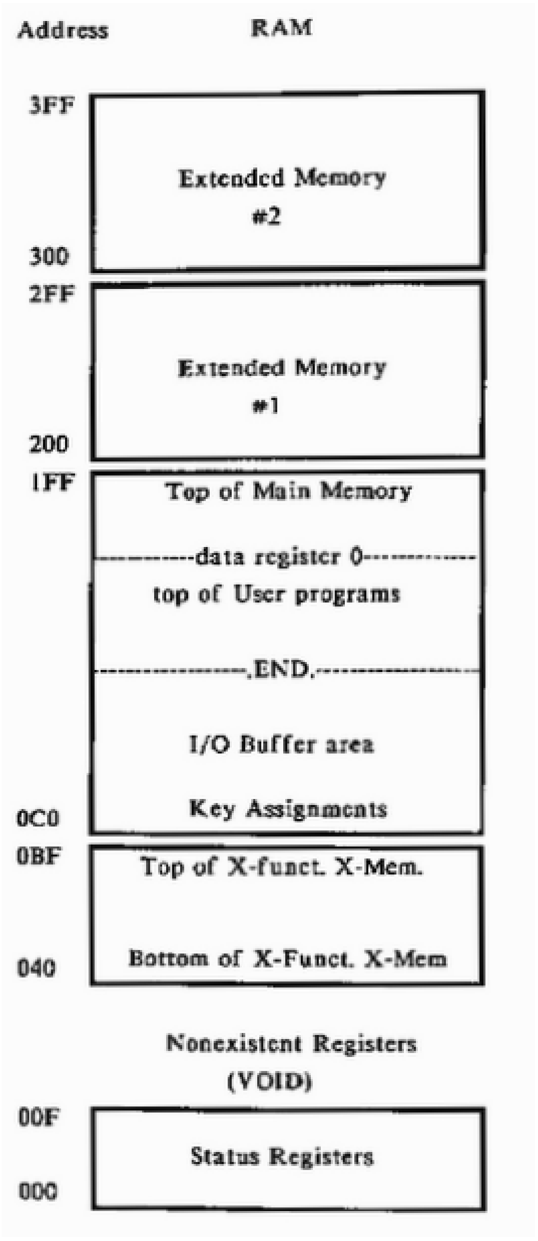
The control keys for RAMEDIT are as follows:

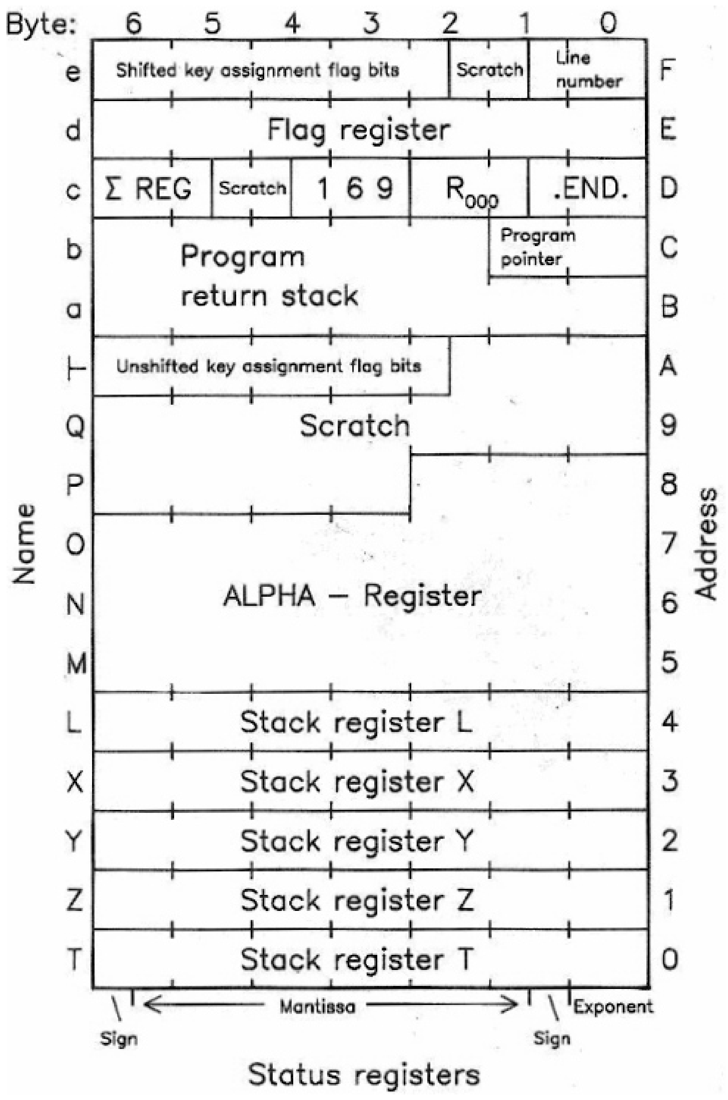
[**USER**]:	moves down to the previous nybble or position within the field
[**PRGM**]:	moves up to the next nybble or position within the field
[**+**]:	moves up to the next register
[**-**]:	moves down to the previous register
[**.**]:	the Radix key moves between both fields, use it to change the register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

A couple of remarks are in order:

- **RAMEDIT** 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 maps below). Be very careful not to alter the contents of those system registers inappropriately to avoid MEMORY LOST or system crashes.

- **RAMEDIT** 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.





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 be thoroughly familiar with the theory and practical applications of synthetic programming.

2.2.2. Editing RAM memory with RAMED. (From the ZENROM)

The RAM Editor from the ZENROM provides an editor function, similar to that of the HP-41CX text file editor ‘**ED**”, which permits review and replacement of any bytes, or optionally insertion of bytes in program memory.

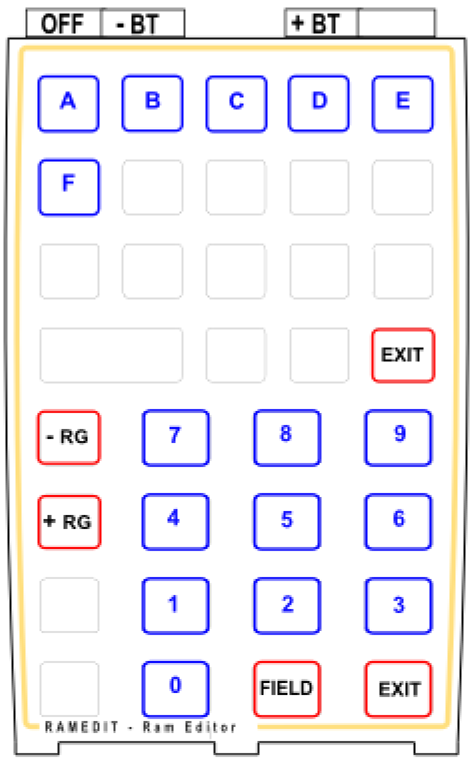
RAMED also redefines the HP-41 keyboard during execution to allow forward or backwards movements through memory in byte or register increments by pressing the [**USER**] and [**PRGM**] keys, (for bytes) and their shifted version for registers.

RAMED takes the start address from status (ALPHA) register M in RUN mode, or from the program counter (status register b) in PRGM mode. If not in PRGM mode, it returns the last reviewed address to M upon exit, or if in PRGM mode, exits at line where it entered (no change to the PC).

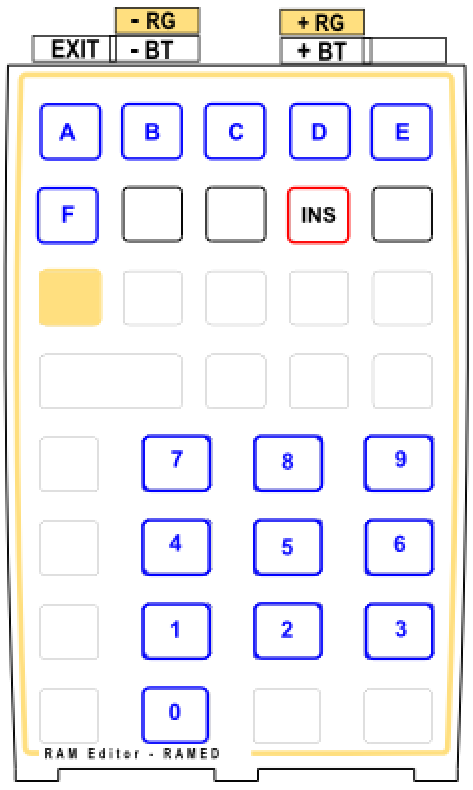
When used inside Program Memory area, pressing the [**I**] key toggles between replace and insert mode – signified by the “1” annunciator being lit in the display. During entry of hexcode values, the back arrow key will cancel the first digit input. By pressing and holding the second digit, the whole hexcode entry is nullified – as it happens during normal HP-41 key-pressing. To exit from **RAMED**, press the [ON] key.

A Quick Comparison.

The figure below compares the redefined keyboards for **RAMEDIT** (on the left) and **RAMED** (on the right). Perhaps the most relevant differences are RAMED’s ability to insert bytes in program mode, and the navigation controls - which in RAMEDIT’s case allow *changing the register* being edited on the fly.



RAMEDIT - GetKey



RAMED - Zenrom

Using RAMED Outside Program Memory

The following section is taken from the ZENROM Manual.

RAMED can prove very useful for examination of memory and system status register structures plus provide the possibility to directly modify or replace their byte contents. For example, you can directly modify the key=assignment information.

To use **RAMED** out of program mode, the starting address is taken from Alpha – more specifically the rightmost four hex-digits of register M, which are the rightmost two characters as seen in the display. By this you can specify the exact register and byte within that register at which you wish to start editing.

This means that if you know the absolute address of the place in HP-41 memory that you want to edit (see the memory map in previous page), then simply use the synthetic text entry feature provided by functions **CODE**, **HEXIN**, or **HEXKB** (any of them will do) followed by STO M. Once the two characters are in ALPHA you can execute RAMED, and you'll be editing memory, starting at the address specified.

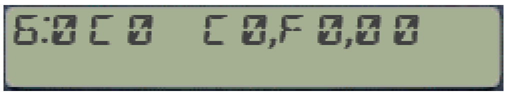
As an example, let's take a look at the key assignments registers, which have a format as follows:

Byte #	6	5	4	3	2	1	0
Bytes:	F0	A7	20	34	04	61	83

Bytes	Description
0	Keycode of key to which assignment is made
1 & 2	Assignment information
3	Keycode of key to which assignment is made
4 & 5	Assignment information
6	Register ID to specify a KA register (F0 hex)

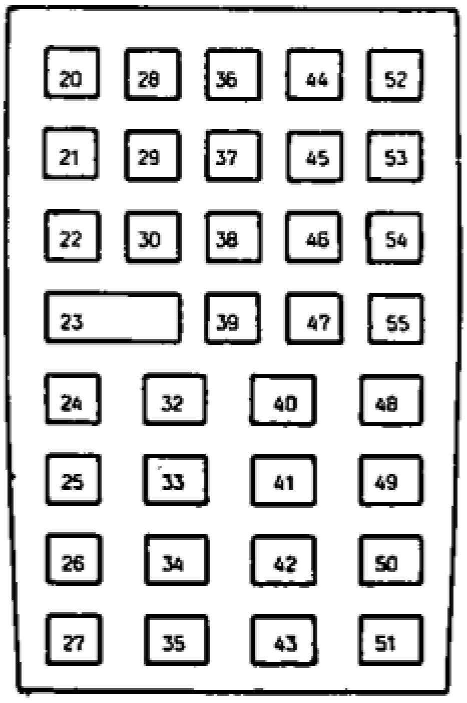
Suppose you wish to edit the lowest key assignment register, which is at address 0C0, and you want to go in at byte 6 of that register (that should contain F0). In standard RAMED notation this is address "6:0C0" – where the ":" character separates the byte from the register address.

To do this, execute **HEXIN** (or **HEXNTRY**) and type "60C0", followed by R/S, STO M. Then execute **RAMED**. Assuming there are no key assignments, the display will now show:


6:0C0 C0,F0,00

You can now begin editing the assignment register. Remember that you will also need to set the key bit-maps in register 10(+) for un-shifted keys, and 15(e) for shifted keys; depending on the assignment.

Note: I don't know you but I always felt a bit shortchanged with this example – which basically doesn't tell you how to edit the key bit-maps. Also the manual refers to another example where there's a circular reference to the status registers structure, so let's include these in here as well.-

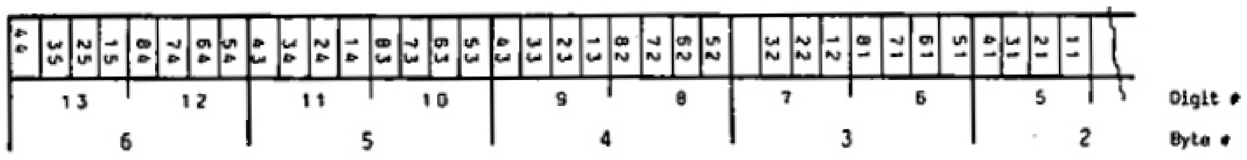


Basically the trick consists of setting the appropriate bits in status register 10 ("|-") and 15 ("e"), depending on whether it's a un-shifted or shifted assignment.

Each bit within those registers represents one key on the keyboard, as per the following mapping – linking the key bitmap on the left with the bit position.

So you'd need to work out which bit needs editing, and come up with the equivalent nybble codes to write on the appropriate status register, using **RAMED** of course.

Far from an automated approach, to say the least, but as they say "with power comes responsibility", and after all **RAMED** is not meant to be used unless you know your way around the system.



Remarks:-

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 be thoroughly familiar 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.

2.3.- EXTENDED MEMORY FNS

This group includes functions related to the X-Memory control and enhanced functionality.

ARCLCHR	ARCL Char from ASCII file	Appends character to ALPHA	Håkan Thörngren
FLHD	File Header	Returns address to X	Ángel Martin
FLCOPY	File Copy	'Source,Destination" in ALPHA	Ángel Martin
FLTYPE	File Type	Gets file type to X	Ángel Martin
GETST	Get Status Registers from File	Status FileName in Alpha	Ángel Martin
READXM	Overwrites all XM from IL File	FileName in ALPHA	Skwid
RENMFL	Rename X-Mem File	"OldName,NewName" in ALPHA	Ángel Martin
RETPFL	Retype X-Mem File	New type in X, FileName in Alpha	Ángel Martin
RSTCHK	Reset Checksum	Program FileName in Alpha	Håkan Thörngren
SAVEST	Saves Status Registers	FileName in ALPHA	Ángel Martin
WORKFL	Gets Work FileName	Appends name to ALPHA	Sebastian Toelg
WRTXM	Writes all XM to IL File	FileName in ALPHA	Skwid
XQXM	Executes a Program File	Program FileName in ALPHA	Ross Wentworth

The appendix-2 has a detailed description of the different X-Mem file header structures, which should help to better understand the functionality provided by these functions. The following short descriptions summarize the most important points for each of them:

- **ARCLCHR** appends the character at the current pointer position of the current ASCII file. The file Pointer is advanced one position, ready to retrieve the next character if needed. Originally published in PPCJ V13 N7 p19

- **FLHD** will return the absolute address (in decimal) for the Header register of the file named in Alpha (or the current file if blank). This is useful as input for **PEEKR** and **POKER**, **RAMED** and other memory editing functions.

- **FLTYPE** returns the type of the file which name is given in Alpha. Valid file types are shown in the table below, note the five custom extensions supported by the AMC_OS/X module:

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

- **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

- **RSTCHK** is a rescue function that restores the checksum value for a PROGRAM file. Use it if this byte gets corrupt or when you alter the program file manually (hacker beware!), so the file will recover its "valid" status. See original article on PPCJ V13 N2 p14

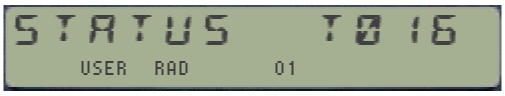
- **XQXM** is a PROGRAM File Execute - direct execution of the program. Note that all GOTO's must be pre-compiled, and no calls to other programs may exist within the file.

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

- **WORKFL** will append the name of the current file (a.k.a the workfile) to ALPHA. Easy does it! This becomes very useful when working with MATRIX files, see the SandMatrix Module if interested.

- **FLCOPY** is a handy utility that allows copying complete like-to-like files of any type. Requires both file names in Alpha, separated by a comma: "***from,to***" (or "NAME ERR" will occur). Both files must exist in X-Memory (or "FL NOT FOUND" will occur), be of the same type (or "FL TYPE ERR" will occur), and have the same size (or "FL SIZE ERR" will occur). The contents of the source file will be copied to the destination. The File Names and Headers will not change.

- **SAVEST** and **GETST** are special in a couple of ways. For starters because their subject is the complete Status Registers, i.e. the "Chip0" of the system RAM. Use **SAVEST** to make backups of the entire status registers area to XMEM, including the stack, flags, Alpha, and the other control registers. Use **GETST** to restore the status registers back to the same state. For obvious reasons the file size will always be 16. They're also special because they use a file type 7, which is properly recognized as type "**T**" by the CAT'4 implementation in the AMCOS/X module:



```
STATUS     T016
  USER  RAD      01
```

  A couple of observations are in order:

  - The X-Mem file name is expected in ALPHA, thus this imposes a little limitation on things. You can however add a comma to the FileName and write additional text after it – which will be ignored by the functions.

  - Register 12(b) stores the program counter (PC). Executing **GETST** in a program will overwrite the current PC, and the program execution will be "lost" – going to the same place it was at when the status registers were saved. There are more tricky issues using these in PRGM mode, like the question of the subroutine stack and the program line. Suffice it to say it's not really advisable – yet I resisted the idea to make it non-programmable, but users beware!

  - Saving and restoring the Key Assignments involves two separate actions. **GETST** only restores the key mappings in registers 10(|-) and 15(e), but it doesn't have anything to do with the actual KA registers in the I/O area. Make sure you use **SAVEKA** and **GETKA** instead for this need, or the key assignments will be scrambled. See the KA utilities section.

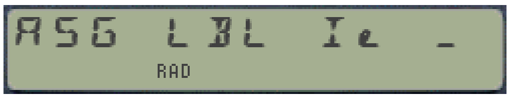
3. RAMPAGE: KA / Buffer Functions

3.1.- KEY ASSIGNMENT FNS

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

ASG _	**Multi-byte ASN**	Supports synthetics	Frits Ferwerda
GETKA	**Gets KA from File**	FileName in Alpha	Håkan Thörngren
KACLR	**Clears KA / Buffers**	OK/OKALL in Alpha	Hajo David
KALNG	**KA Length**	No inputs	Hajo David
KAPCK	**Packs KA Area**	No inputs	Hajo David
KYOFF	**Deletes KA from Key**	Prompts for Key	Frits Ferwerda
LKAOFF	**Deactivates Local KA**	No inputs	Gordon Pegue
LKAON	**Activates Local KA**	No inputs	HP Co.
MRGKA	**Merge KA to File**	FileName in Alpha	Håkan Thörngren
SAVEKA	**Save KA to File**	FileNAme in Alpha	Håkan Thörngren
VKEYS	**View Keys**	Enumerates KA	PANAME ROM

- **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 RAMPAGE calling it. You're encouraged to refer to the MLROM documentation for further details.



```
ASG LBL Ie _
    RAD
```

Note that **ASG** turns the ALPHA mode ON automatically upon execution, so there's no need to press it twice – this is an improvement over the standard HP-41 OS behavior, that now can be used across all Library#4-aware modules.

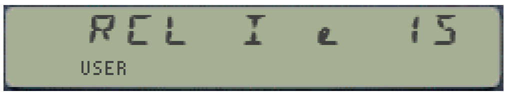
Example:- Assign the synthetic function 'RCL IND e" to the LN key.

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

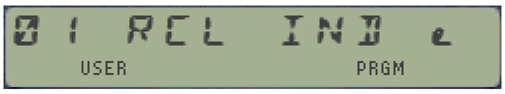
Alternatively you can execute the **ASG** function and spell out the function name. Note that the ALPHA mode is turned on automatically for this:

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

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



```
RCL Ie 15
USER
```



```
01 RCL IND e
USER         PRGM
```

- **KACLR** is used to erase all Key Assignments and possibly also the buffers. It expects a confirmation string in Alpha. With "OK" only the KA will be erased (so equivalent to **CLKEYS** in the CX); but with "OKALL" the complete IO area will be purged (that is the KA registers and ALL buffers).

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

- Saving and getting KA in/from Extended Memory with **SAVEKA**, **GETKA** and **MRGKA** also expect 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..

- **LKAON** and **LKAOFF** are meant to be used together, to temporarily suspend the local key assignments (on keys A-J) so that it doesn't interfere with local program labels. These functions only modify the key mappings in status registers 10("|-") and 15("e"), not altering the actual KA registers.

- **KAPCK** will pack the KA registers, compacting the voids (blanks) left behind when un-assigning individual function keys. The diagram below shows that each KA register can hold up to <u>two</u> 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

- **KYOFF** is a selective KA removal function. It prompts for the key you wish to remove any assignments from, either un-shifted or shifted. Upon completion the assignment is gone entirely, i.e. cannot be "reactivated". This function is taken from the ML ROM, compiled by Firtz Ferwerda and Meindert Kuipers.

- **VKEYS** is a KA Catalog alternative to the CX or the OS/X implementations. It shows a sequential list of all key assignments in the system, making short pause in-between entries. No control characters (or hot keys) are available. The output will be sent to a printer is connected and set in NOR/TRACE modes. No input is required. **VKEYS** is taken from the PANAME ROM, therefore credited to Stephane Barizienne. It is an MCODE equivalent to the PPC routine '**VK**", in case you remember.

3.2.- BUFFER FUNCTIONS

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

BFCAT	**Buffer Catalog**	Enumerates Buffers	Ángel Martin
BFLST	**Lists Buffers**	Shows list in Alpha	Ángel Martin
BFLNG	**Buffer Length**	Buffer id# in X	David Yerka
BUFHD	**Buffer Header**	Displays all registers	Ángel Martin
BFHEAD	**Decodes buffer header**	Shows decoded header	Ángel Martin
CLBUF	**Clears Buffer contents**	Zeroes all data	Ángel Martin
CRBUF	**Creates Buffer**	Address to X	Ángel Martin
DELBUF	**Deletes Buffer**	ID#,SIZE in X	Ángel Martin
GETBF	**Gets buffer from File**	Buffer id# in X	Håkan Thörngren
SAVEBF	**Save Buffer to File**	FileName in Alpha, id in X	Håkan Thörngren
REIDBF	**Re-issue Buffer id#**	OLDID,NEWID in X	Ángel Martin
RESZBF	**Resize Buffer**	ID#,SIZE in X	Ángel Martin

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.

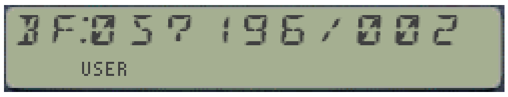
2. Each buffer has a header register (at the bottom) that holds its control data. The structure of the header varies slightly for each buffer, but all must have the buffer type (a digit between 1 and E) repeated twice in nybbles 13 and 12, as well as the buffer size in nybbles 11-10 (maximum 0xFF = 255 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. The screen below shows an example of an “orphan” buffer id#5 on V41:


```
BF:05? 196/002
USER
```

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 **BFLNG** and **DELBUF**. The X-MEM Save/Get functions **SAVEBF** and **GETBF** also expect the File Name in Alpha.

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.

3.2.1 Creating, Retyping and Resizing Buffers.

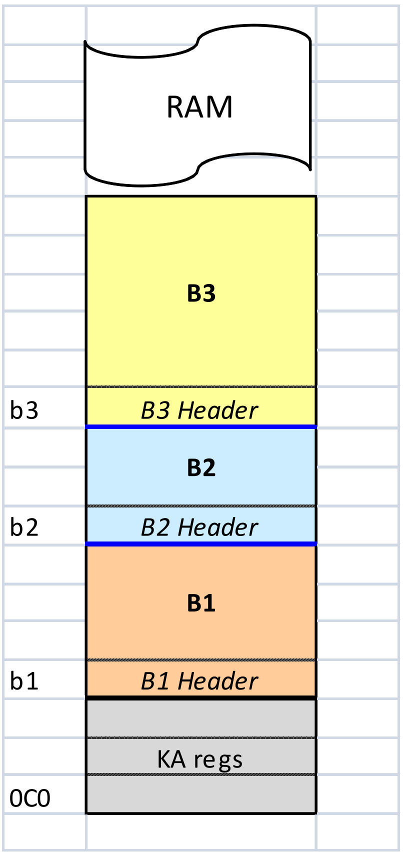
The input for **CRBUF** and **RESZBUF** is given in X as a combined decimal number: ID#,SIZE. The integer part represents the buffer id# (must be between 1 and 14), and the decimal part its size in registers, including the header (must have three decimal places). Maximum size is 255 registers; larger values (as well as zero) will trigger “DATA ERROR” messages.

This convention also applies to **REIDBF**, so the new id# must be expressed with three significant digits. For example, to change buffer #9 into buffer #10 we type: **9,010** ; XEQ “**REIDBF**”

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

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

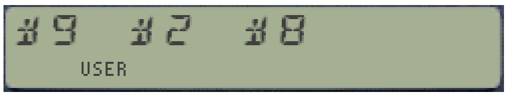
CRBUF, RESZBF as well as **GETBF** will check for available memory in the RAM I/O area, showing “NO ROOM” when there isn’t enough room RAM to proceed.



3.2.2. General Buffer Utilities.

The section header **–KA/BUF FNS** is a “stealth” access for **BUF?**, which checks 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.

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



```
#9  #2  #8
  USER
```

CLBUF clears the buffer contents (i.e. zeroes all registers) but does not delete the buffer. The buffer is still available and will show in the buffer catalog.

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

BUFH and **BFHEAD** are related functions. **BUFHD** will recall (as a NNN) the contents of the header which id# is specified in X. **BFHEAD** goes one step further and will automatically decode the contents into its HEX equivalent. It will be placed in ALHA. You can think of **BFHEAD** as the combination of **BUFHD** and **DCD** together.

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").

3.2.3. Buffer Data Exchange functions

This function group deals with data transfer between Buffers and Main memory.

ARCLBF	**Appends buffer to ALPHA**	Buf id# in prompt	Ángel Martin
ASTOBF	**Stores ALPHA in Buffer**	Buf id# in prompt	Ángel Martin
BF<>RGX	**Swaps w/ Data Registers**	Id# in prompt, RG# in X	Ángel Martin
BF>ST	**Dumps Buffer to Stack**	Buf id# in prompt	Ángel Martin
BFVIEW	**Views buffer contents**	Buf id# in prompt	Ángel Martin
RGX>BF	**Copies Data Regs to Buf**	Id# in prompt, RG# in X	Ángel Martin
ST>BF	**Copies Stack to Buffes**	Buf id# in prompt	Ángel Martin

Pretty obvious in their scope, with almost self-explanatory names. They all use a prompt to input the buffer id#, and the X-register should have the first register of the block when data registers are involved (not needed for ALPHA or Stack destinations).

In addition to the usual "NO BUF", and "DUP BUF" messages - if the buffer is longer/shorter than the source/destination, only the fitting registers will be transferred and an "END OF BUF" message will be shown. Register existence will be performed as well, returning the "NONEXISTENT" error if not available. You know what to do.

They pretty much cover all contingencies, except perhaps the **BF>RGX** case. You can use the sequence of **BF<>RGX** and **RGX>BF** together for that case.

But we're not done yet: in the tradition of leaving the best for last, let's see the Buffer Catalog to end this section.

3.2.4. Buffer Catalog (*)

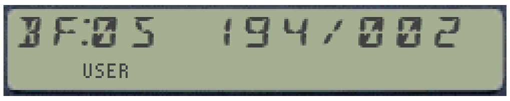
BFCAT	**Buffer CATalog**	Hot keys: R/S, SST, SHIFT, D, H	Ángel Martin
[D]	Deletes Buffer	*In manual mode*	Asks Y/N?
[H]	Decodes Header register	*In manual mode*	

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.

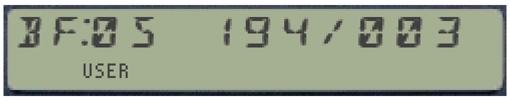
Each buffer has an id# number, ranging from 1 to 14. Only one buffer of a given id# can exist, thus the maximum number present at a given time is 14 buffers – assuming such hoarding modules would exit – which thankfully they don't.

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.


BF:05   194/002
USER

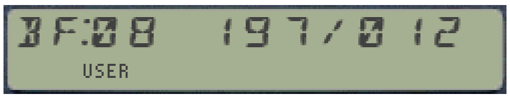
id#=5, buffer at address 194, size=2, properly allocated.

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:


BF:05   194/003
USER

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:


BF:08   197/012
USER

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. Other HP modules use them to temporarily store scratch data, like in the HP-IL Development Module for the IL frame codes (functions **MONITOR** and **SCOPE**); or the Advantage Pac as scratch area for calculations (used in functions **SOLVE** and **INTEG**).

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	Time Seed, Last Function data
10	Time Module	Alarms Information
11	Plotter Module	Data and Barcode parameters
12	IL-Development; CMT-200	IL Buffer and Monitoring
13	CMT-300, FORTH-41	Status Info, FORTH Code
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.

(*) CATalog functions are notoriously complex and take up a significant amount of ROM 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.

4. TOOLBOX: System Information Utils.

4.1. SYSTEM INFO UTILS

The first group of functions in the TOOLBOX can be described as "General Information" utilities. Use them to retrieve system set-up and configuration details.

ΣRG?	**Stat Regs Finder**	Stat Regs Address in X	Ken Emery
CRTN?	**Curtain Finder**	Curtain Address in X	Nelson F. Crowle
DREG?	**Data Registers Finder**	Current Size	Ken Emery
DSP _	**Sets decimal places**	Input dec# in prompt	Sebastian Toleg
DSP?	**Decimal Places Finder**	decimal places in X	Ángel Martin
FREG?	**Free Registers Finder**	Main Memory Regs	Ken Emery

- **ΣRG?** Returns the current location of the Statistical Registers - i.e. those used by the Statistical functions to accumulate the data pairs. Basically it's identical to the CX function **ΣREG?**.

- **CRTN?** Returns to X the absolute address of the curtain (i.e. separation between program and data registers). It is also the absolute address of data register R00. No input value is required.

- **FREG?** Returns to X the number of available (free) program registers in Main Memory. No input value is required. The general equation is: Total Registers = (Data + Program) Regs; where Total Regs = 512 on the CV and CX models.

- **DREG?** Is another SIZE finder, slightly faster than the native version **SIZE?**

- **DSP?** is used to return the number of decimal places currently selected in the display. This is independent from the decimal mode, FIX / SCI / or ENG.

- **DSP** is the reverse function: use it to specify the number of decimal places shown in the display, between 0 and 9. Input the number in the prompt or in the X-register if used in a program – all without changing the FIX/SCI/ENG mode. The prompt will be maintained is an invalid input is entered.

Note that the storage location for the details retrieved by some of these functions is the status register 13(c), as per the table below:

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

Whereas the information for the number of decimal places to display (from zero to 15 theoretically, although the display only has 10 physical positions) is held in the flags register, specifically flags 36 to 39 (see table on next page for details).

4.1.2. Flag Handling functions.-

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

CFX _ _	**Extended CF**	Allows ANY flag#	Michael Katz
SFX _ _	**Extended SF**	Allows ANY flag#	Michael Katz
TGFX _ _	**Toggle Flag**	Allows ANY flag#	Ken Emery
FC?S _ _	**Is Flag Clear and Set it**	Allows ANY flag#	Ken Emery
FS?S _ _	**Is Flag Set and Set it**	Allows ANY flag#	Ken Emery

- **CFX** and **SFX** are natural extensions to the mainframe standard Clear/Set Flag functions. Unlike those, the input is expected in X as a decimal entry. Also unlike them they'll accept anyone of the 56 flags from 0 to 55. When used in a program enter the flag# in the preceding program line.

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

- **FS?S** and **FS?C** are the symmetric counterparts to the native **FC?S** and **FC?C** functions. They extend the logic and provide shorter handling – sometimes not possible without them. They also operate on the complete flag range, as you'd expect.

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

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

Note. As a sideline, the example below illustrates an unexpected application of **X<>F** and **X<>BM** – applied to the leftmost byte of the status register 14(d), i.e. the user flags 0-7. Basically these functions can be used to invert the bits in the byte, as their results are logically reversed values.

- First get the address in ALPHA, ZENROM-style: **HEXIN** "600E", R/S, STO M
- Now to obtain the NOT of the value in X, type: **X<>F,** XEQ "**X<>BM**"

4.1.3. HP-IL Related Functions

The Extended Functions module gave us GETAS and SAVEAS to write and read ASCII files to HP-IL Mass Storage devices, but nothing about DATA files. This gap is now closed by the functions described below.

FSIZE?	**HPIL Media File Size**	FileName in ALPHA	R. del Tondo
READF	**Read Data File**	IL FName, XM FName	R.del Tondo
WRTDF	**Write Data File**	XM FName, IL FName	R. del Tondo
READPG	**Reads page from HP-IL**	Page# and FileName	In the OS/X Module
WRTPG	**Writes page to HP-IL**	Page# and FileName	In the OS/X Module

- **FSIZE?** Returns to X the length in registers of the (primary) mass storage file which name is specified in Alpha. If no HP-IL is present on the system the error message “NO HPIL” will be shown.

- **READF** and **WRTDF** are used to read and write individual DATA files between the IL Drive and XMEM. To use them properly you need to first create the destination files (like **GETAS** and **SAVEAS** do for ASCII file types).

  Fortunately you can use **FSIZE?** And **FLSZE** to find out that required piece of information, and then create the file appropriately either in X-Mem or in the Mass Storage device. The FOCAL programs below would do that automatically – just type the source and destination file names in ALPHA separated by a comma:

```
01  LBL "GETDF"          08  LBL "SAVEDF"
02  FSIZE?               09  FLSZE
03  ASWAP                10  ASWAP
04  CRFLD                11  CREATE
05  ASWAP                12  ASWAP
06  READF                13  WRTDF
07  RTN                  14  END
```

Note: These functions are related to **READXM** and **WRTXM** seen before, but remember that those operate on the whole XMEM contents, not on individual files.

(*) The function **ASWAP** is available in the AMC_OS/X Module, The ALPHA_ROM, and the PowerCL Module amongst other sources.

4.1.4. Other Miscellaneous Utilities

The rest of this pack - not necessarily related but grouped just the same.

APPEND _	**Append to ALPHA**	Appends prompt as text	W. Doug Wilder
BST^	**Repeat BST**	Repeated while pressed	Nelson F. Crowle
SST^	**Repeat SST**	Repeated while pressed	Nelson F. Crowle
CSST	**Continuous SST**	Automated PRGM listing	Phi Trinh
CVIEW	**Continuous VIEW**	Non-halting AVIEW	Frits Ferwerda
GTEND	**Go to .END.**	Sets PC to .END.	Ken Emery
LASTP	**Goes to Last Program**	Sets PC to begin of PRGM	ZENROM
NOP	**No Operation**	Inserts “F0” in program	ZENROM
TGPRV _	**Toggle PRIVATE status**	Program Name in Alpha	Sebastian Toelg
PC<>RTN	**Exchanges PC and RTN Adr**	Alters PRGM execution	W&W GmbH
XQ>GO	**Pops Return Address**	One level is removed	Håkan Thörngren
XROM _ _:_ _	**Executes XROM Function**	Prompts ROM and FCN id#	Clifford Stern
RTN?	**Pending RTN stack**	YES/NO, skip if false	W. Doug Wilder

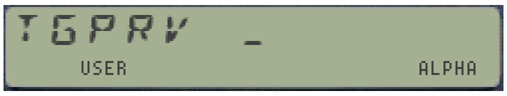
- **CVIEW** is a non-stop AVIEW, so it avoids the printer halt even if flag 21 is set.

- **GTEND** Sends the program counter to the permanent .END. in program memory (the position of the Curtain). Almost identical to it is **LASTP**, the difference being that the program pointer is placed at the first line of the Last program in RAM, i.e. the one closest to the .END.

- **NOP** is a useful No-Operation function. While in RUN mode it pretty much useless, when used in a program it'll mutate into a synthetic Text-0 line (“F0” byte) – which is the fastest and smallest NOP known to man in a FOCAL program. Seeing is believing… try it for yourself.

- **APPEND** is equivalent to using the “append” functionality manually, i.e. pressing [**ALPHA**], and “**|-**“ keys. More interestingly, in a program it will prompt the existing ALPHA contents for the user to continue entering more. Similar to **PMTA**, but the initial string will remain.

- **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 MLDL/HEPAX RAM (seen as ROM by the calculator). Note that in RUN mode, the ALPHA mode is switched on automatically for you to spell the program name. **TGPRV** is also programmable. If Alpha is empty the program pointer must be set to any line within the program.



- **XQ>GO** is a nifty function that drops one RTN address from the subroutine RTN stack. This can come very handy when you don't want to return to the calling XEQ while running a subroutine, for example depending on the partial results obtained.

Note: You can combine this function with **RTN?** (Stealth in the module, using the section header “-HACKER LAB”) if you want to remove all pending RTN addresses from the RTN stack, using the following FOCAL routine:

```
01  LBL”POPALL”          04  RTN?  (-HACKER LAB)
02  LBL 00               05  GTO 00
03  XQ>GO                06  END
```

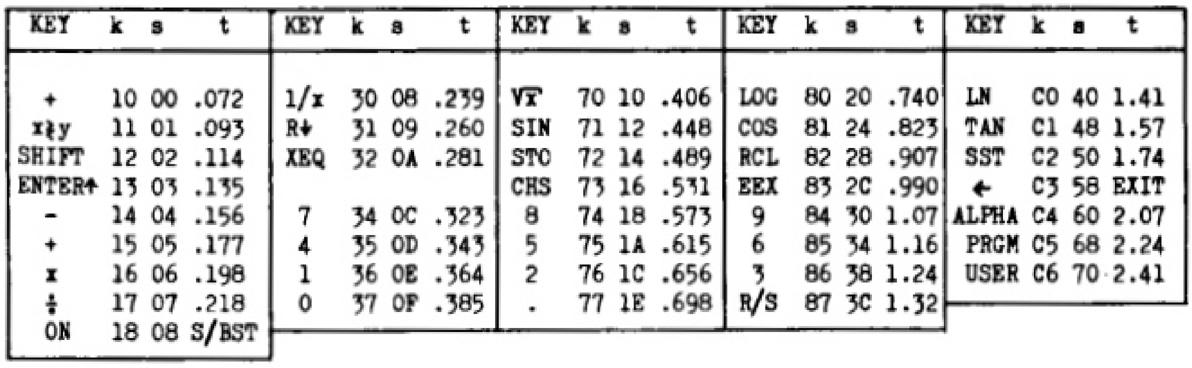
- **SST^** and **BST^** are lazy-users (and keyboard-savers!) methods to advance or go back in a FOCAL program in a semi-automated fashion: the SST/BST action will be repeated while the key is depressed, with a small pause between each program line displayed. You must be in PROG mode for these functions to work as intended.

- **CSST** is a more automated approach to the same task. It sequentially displays the program steps of the program pointed at by the Program Counter (PC). It's equivalent to using the **SST** key multiple times, and thus its name.

  This function is programmable, operating in single-step mode or in back-step mode depending on whether the user flag 0 is cleared or set. The back-arrow key terminates the display of program lines, yielding to normal keyboard operation in RUN mode, or transferring control to the running FOCAL program that executed **CSST.** In the latter case, the program execution resumes with the currently displayed line (i.e. it has moved the program pointer).

  The [**ON**] key switches between single-step operation to back-step operation and vice-versa (i.e. toggles between both modes). The "**0**" annunciator is visible in the display whenever back-step operation is in effect.

  The delay between lines shown can be adjusted by pressing any keyboard key, see the table below taken from the original article in PPCJ V9N7 p49 (refer there for further details). To use it, position first the PC at the target location (using GTO or similar). Note that the display time increases linearly from the top key down to the bottom key in a given key column on the keyboard, and continues to increase on the next column to the right. Furthermore (staying on the same row of keys), pressing the key one column right of the selected key will roughly double the selected display time.



KEY	k	s	t	KEY	k	s	t	KEY	k	s	t	KEY	k	s	t	KEY	k	s	t
+	10	00	.072	1/x	30	08	.239	√x	70	10	.406	LOG	80	20	.740	LN	C0	40	1.41
x⇄y	11	01	.093	R↓	31	09	.260	SIN	71	12	.448	COS	81	24	.823	TAN	C1	48	1.57
SHIFT	12	02	.114	XEQ	32	0A	.281	STO	72	14	.489	RCL	82	28	.907	SST	C2	50	1.74
ENTER↑	13	03	.135					CHS	73	16	.531	EEX	83	2C	.990	←	C3	58	EXIT
−	14	04	.156	7	34	0C	.323	8	74	18	.573	9	84	30	1.07	ALPHA	C4	60	2.07
+	15	05	.177	4	35	0D	.343	5	75	1A	.615	6	85	34	1.16	PRGM	C5	68	2.24
x	16	06	.198	1	36	0E	.364	2	76	1C	.656	3	86	38	1.24	USER	C6	70	2.41
÷	17	07	.218	0	37	0F	.385	.	77	1E	.698	R/S	87	3C	1.32				
ON	18	08	S/BST																

CSST SPEED TABLE

k: key code
s: speed code (HEX)
t: display time, in seconds per line, for a RAM program line

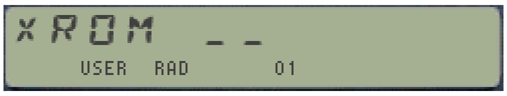
There are 37 different speeds available at the touch of a single key. The default speed may be reselected at any time by pressing the [SQRT] key in the middle column. Holding down any key (except the back-arrow) freezes the display at the current program line. The user is therefore given complete control over the speed and direction of the flow of program lines on the LCD display. Note that it won't list PRIVATE programs… but of course you know how to overcome that, right?

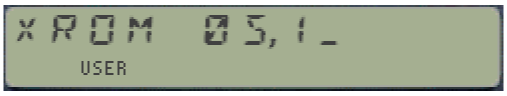
PRIVATE
USER

- **RTN?** Is included as stealth under the section header function "-HACKER LAB". It looks at the RTN stack checking for pending levels, reporting YES/NO depending on the case. If used in a program it'll skip the next line if false.

  Note: **PC<>RTN** in the AMC_OS/X Module can be used in combination with **RTN?**. It 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).

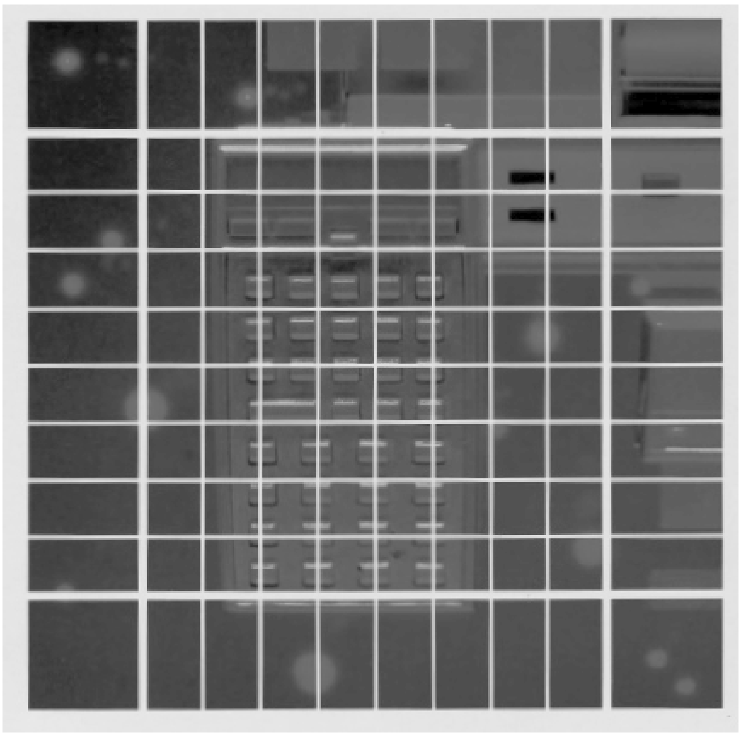
- **XROM** is a well-known function to directly call any function within a plug-in ROM, knowing its ROM id# and function#. Written by Clifford Stern in the heydays of the 41 systems, with a real inside knowledge of the internal OS routines [PARSE]. Both prompt inputs are to be entered as DECIMAL values.


```
XROM  _ _
   USER  RAD      01
```


```
XROM   05,1_
   USER
```

  This is helpful to program functions even if the module they reside in is not present – but obviously will require it to be plugged when the function is to be executed.

  Another unusual aspect of this function is that it can be coaxed to access internal (mainframe) functions and routines – using zero as ROM id#. There was a slew of articles along the years on the "XROM 00,xx" subject, feel free to dive into the PPC vault and fish for related topics.



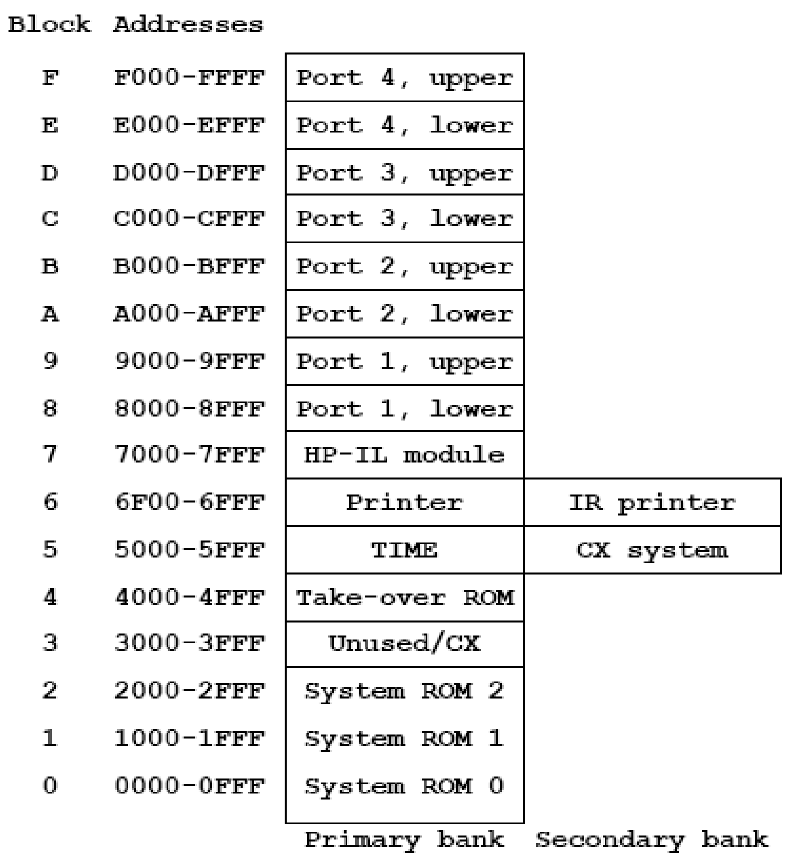
4.2. SYSTEM & PAGES

4.2.1. Introduction: I/O Bus , Ports and Pages.-

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.

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



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

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**; quiite an impressive feat considering we’re talking about a hand-held calculator design from 1979 – which although extended, expanded, and stretched to the limit really shows the versatility and solid engineering of the design.

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

4.2.2. The system as a whole.

CHKSYS	**Checks ROMS plugged in**	OK/BAD	Ángel Martin
ROMLST	**Lists ROMS plugged in**	String of ROM id#'s	Ángel Martin
OSREV	**Shows OS Revisions**	String with rev's	Nelson F. Crowle
PGCAT	**Page Catalog**	VM Electronics	Source: HEPAX Module
ROMCAT _ _	**ROM Catalog**	ROM id# in prompt	J.D. Dodin
PTCAT _	**Port Catalog**	Port# in prompt	Nelson F. Crowle
IOBUS _	**I/O Bus Usage**	Free, Used, Banked	Ángel Martin

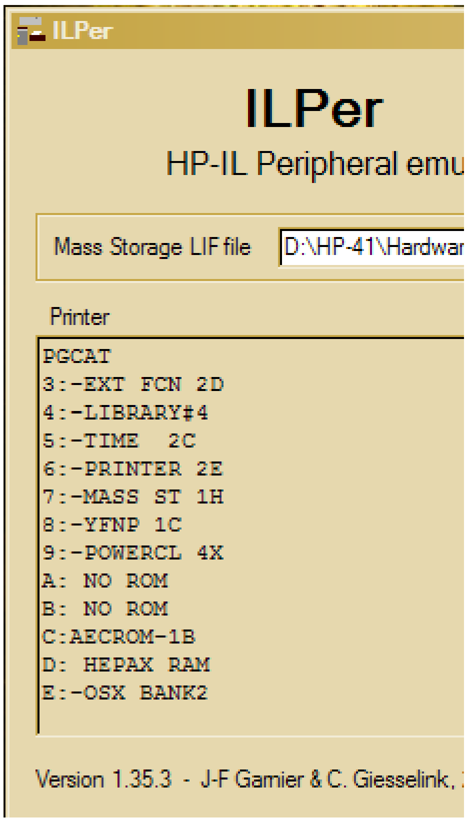
PGCAT is taken from the HEPAX Module (called **BCAT** there, within the HEPAX sub-functions group) -and written by Steen Petersen. **PGCAT** enumerates the first function of each page, starting with page 3. The enumeration can be stalled pressing any key other than R/S or ON, but the individual functions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

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

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

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

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



ILPer

ILPer

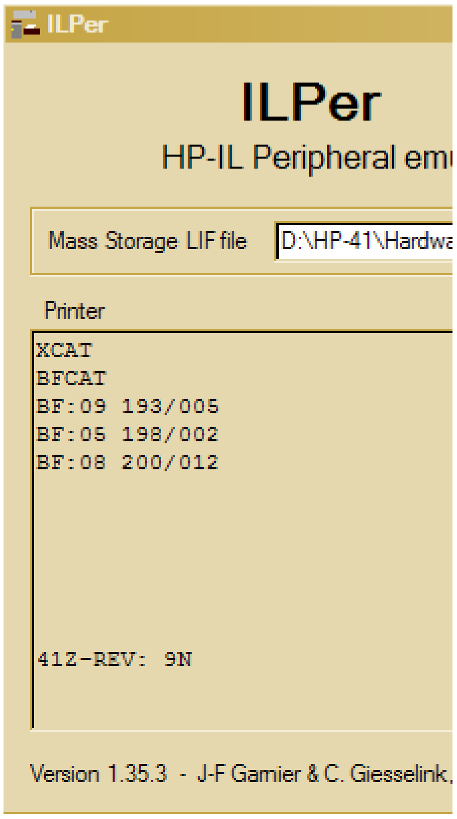
HP-IL Peripheral emu

Mass Storage LIF file D:\HP-41\Hardwar

Printer

```
PGCAT
3:-EXT FCN 2D
4:-LIBRARY#4
5:-TIME  2C
6:-PRINTER 2E
7:-MASS ST 1H
8:-YFNP 1C
9:-POWERCL 4X
A: NO ROM
B: NO ROM
C:AECROM-1B
D: HEPAX RAM
E:-OSX BANK2
```

Version 1.35.3 - J-F Garnier & C. Giesselink,



ILPer

ILPer

HP-IL Peripheral emu

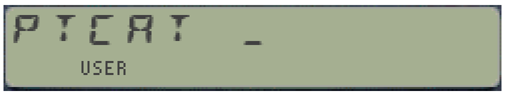
Mass Storage LIF file D:\HP-41\Hardwa

Printer

```
XCAT
BFCAT
BF:09 193/005
BF:05 198/002
BF:08 200/012






41Z-REV: 9N
```

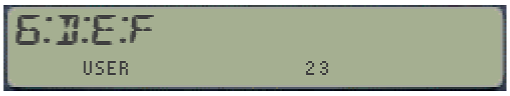
Version 1.35.3 - J-F Garnier & C. Giesselink,

- **PTCAT** is related to the same concept, but driven by the port number instead of the pages. The CAT 2 enumeration will start at the port number input at the prompt, which is expected to be 1,2,3 or 4. In fact the prompt <u>will be maintained</u> if a value greater than 4 is input (a more intelligent error handling method than putting up an error message like “DATA ERROR”). Entering zero will start at Page#5, listing the internal modules as well.



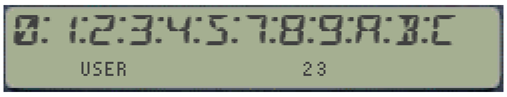
```
PTCAT _
USER
```

- **IOBUS** will present colon-separated strings of hex numbers corresponding to those free, used, or bank-switched pages in the calculator – according to the input value 0,1,2 respectively. Obviously the OS will always be included in the “Used” string, which is a nice clue to quickly tell which particular string you’re looking at. See for instance the examples bellow showing a pretty decent configuration:



```
6:B:E:F
USER          23
```

for the free pages, and

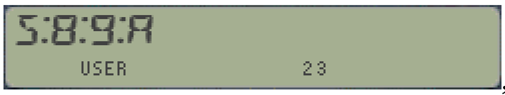


```
0: 1:2:3:4:5:7:8:9:A:B:C
USER          23
```

for the used pages.

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

The bank-switched (or simply “Banked”) corresponds to those pages with a bank-switched configuration, as defined in the ROM signature characters. The official convention is not strictly followed by the (very few) authors of the few bank-switched ROMs, but the number of banks should be marked in characters 2/3/4 of the ROM signature. An example with both the PowerCL and the SandMath_2x2 plugged returns the following:- Can you explain the presence of the “5”? Hurry, time’s ticking out!



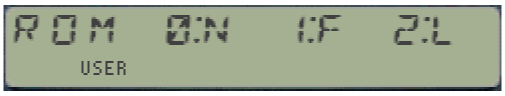
```
5:8:9:A
USER          23
```

,

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

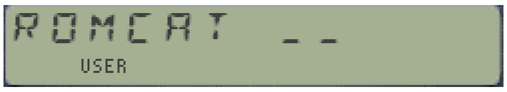
The TOOLBOX implementation is a “little brother” of the function with the same name in the POWER_CL Module. Here the prompt is expecting values 0,1, or 2 as only valid entries. Any other number will simply be ignored and the prompt will be maintained - which is a more intelligent error handling than putting out a NONEXISTENT message.

- **OSREV** simply shows the revisions for the three first pages, containing the core Operating System code (in ROMS 0/1/2) / which for an unmodified HP-41CX are as follows: (Note that this result must be obtained using function **PGSIG,** covered in the next section).



```
ROM   0:N   1:F   2:L
USER
```

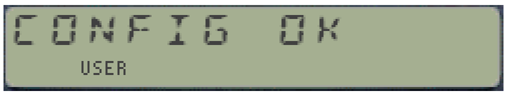
- **ROMCAT** was written by Jean-Daniel Dodin, well-known MCODE pioneer and HP-41 old hand. It prompts for the XROM id# and starts CAT’2 from the position used by such ROM (if present). The usual conventions apply, whereby only the ROM headers are listed unless the catalog is stopped and ENTER^ is pressed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

  
  ROMCAT _ _
  USER

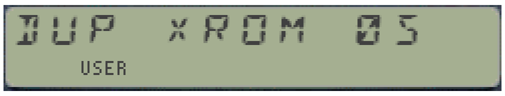
  This is especially useful when the ROM in question is configured in the internal pages of the I/O Bus, like the printer, Time Module or HP-IL interface. You need to know its ROM id# instead, entered as a decimal number in the prompt = or in the X-reg. when used in a program.

- **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.

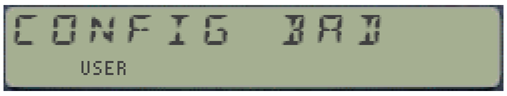
  It will also report all and every offending id# in case of conflicts, as many as there may exist. Use it as frequently as you need, it’s the best way to ensure that things are fine after plugging any of the many modules available on the CL library – a match made in heaven.

  
  CONFIG OK
  USER

  or

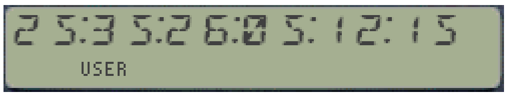
  
  DUP XROM 05
  USER

  plus:

  
  CONFIG BAD
  USER

- **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?

  
  25:35:26:05:12:15
  USER

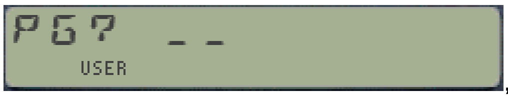
4.2.3 The Pages within.

SUMPG _	**Sums Page checksum**	Page# in prompt/X	George Ioannou
PG? _	**Page vital constants**	Page# in prompt/X	W&W GmbH
ROM? _ _	**Rom vital constants**	XROM id# in X	W&W GmbH
CHKROM _ _	**Verifies ROM checksum**	XROM id# in prompt	HP Co.
CPYPG _ _	**Copies Pages**	Source in X, Dest. in prompt	Ángel Martin
CLBL	**Clear Block**	BBBB|EEEE in X as NNN	Frits Ferwerda
PGSIG _ _	**Page Signature**	PG# in prompt	Ángel Martin
BLANK?	**Tests for blank page**	YES.NO, skips if false	Ángel Martin
PGROOM _ _	**Counts the Page blanks**	Pg# in prompt or X if PRGM	Ángel Martin

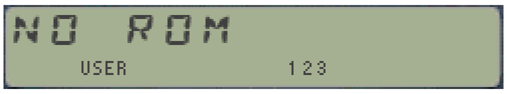
- **PG?** This function returns miscellaneous information corresponding to the page number input in the prompt (as a two-digit decimal format) in RUN mode, or in X as decimal value if run in a program.
  The information returned is as follows:

  - Header function name in ALPHA, and:
  - [XROM id#] ; [# of functions] in X. (in integer and fractional parts)

  An input larger than 15 will cause a NONEXISTENT error message to be shown. If there's nothing plugged in the page the message "NO ROM" will be shown.



Input prompt

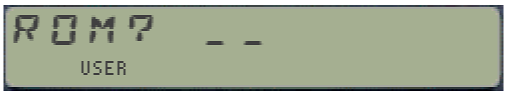


Page is not used (Free).

- **ROM?** is also a prompting function. It returns the ROM vital constants for the XROM id# value input in the prompt (in decimal), as follows:

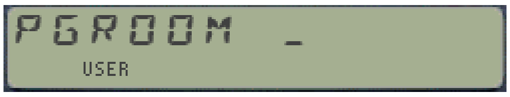
  - Page# where is plugged in X, and
  - number of functions in Y.

  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 (i.e. following the "do if true, skip if false" rule).



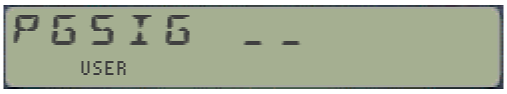
lbl

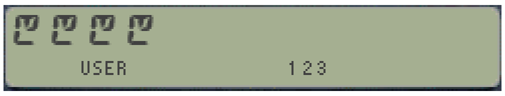
- **PGROOM** counts the number of words with zero value in the page which number is input in the prompt (or in X in PRGM). Interesting to see the density of your favorite MCODE modules (use the OS as a ranking benchmark), and to get an idea on how much room is still available in the page.



Note that if the page is blank the result will be zero – as a proxy for 4,096 words.

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


PGSIG _ _
USER

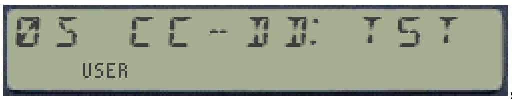

@@@@
USER 123

input prompt

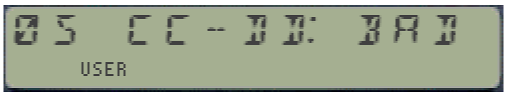
represents a "blank" signature value.

If the input is greater than 15 the functions will show the OS Revisions for ROMS 0,1 and 2 – i.e. the same as function OSREV. This is better than a static "DATA ERROR" message.

- **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.

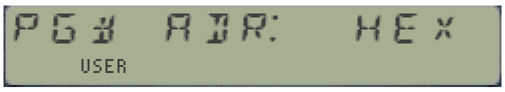

05 CC-DD: TST
USER

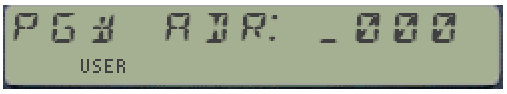
,


05 CC-DD: BAD
USER

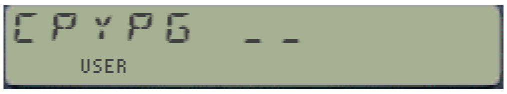
Incidentally it's more than likely that if you run **CHKROM** on the TOOLBOX or RAMPAGE the result is "BAD". This is not because of an error; I just usually don't bother to update the checksum values, as the code is updated very frequently.

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. Needless to say that the checksum won't be written if the page holds a ROM module, and not a MLDL-RAM / HEPAX RAM setup.


PG# ADR: HEX
USER


PG# ADR: _000
USER

- **CPYPG** copies the contents from page number in X to the page entered in the prompt (in decimal format). The destination page must also be RAM-mapped. No confirmation string is required! **CPYPG** is <u>not</u> programmable.


CPYPG _ _
USER

COPYROM in the HEPAX module section, pretty much does the same as **CPYPG**, but taking the FROM: page input from Y and the TO: page input from the X register instead of the prompts. The function name is somehow misleading, as it's operating on PAGE numbers and not on XROM id#'sY. Note that if you use this function on HEPAX RAM pages, the protected status will be ignored – you're instructed to use **COPYROM** instead.

- **CLBL** will clear the block between addresses "bbbb|nnnn" given as a NNN placed in X, which is used as input. <u>If the input is just one digit it'll delete the complete page</u>. Obviously will only work with ROM-mapped pages. Note that **CLRAM** in the "-HEPAXA" section also clears the complete page, and it takes a decimal input for the PG# in X. Both require the string "OK" in Alpha, as a security measure to avoid accidental usage.

5. TOOLBOX - Hacker's Lab

5.1. ROM EDITOR (MLDL)

Editing ROM areas with ROMED.

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

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

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

Here's the original description:

Write ROM. MLDL Q-RAM editor. Execute and supply hex address on the hex keyboard. Use SST, BST and TAN to navigate, press backarrow to enter a new address and backarrow again to exit. Input a new value by keying it in on the hex keyboard, the first digit must be zero to three. The input may be terminated at the last digit by nulling out the keypress. After keying in the new value, the address is automatically incremented. This function uses only the WROM (040H) instruction; it uses the Q register for address storage during low power (partial key sequence) keyboard driver calls. If the word cannot be written, "ROM" is displayed for 300 ms.

Note: This function clears F18 "Interrupt Enable" to prevent the HP-IL Development ROM from destroying Q and the CPU return stack during low power keyboard call (Light Sleep).

ROMEDIT Control Keys:

The operation of ROMED can be controlled with the following keys:

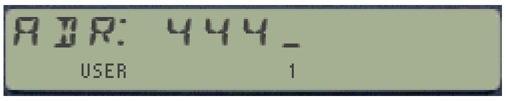
[**SST**] :	moves up one word
[**BST**], [**TAN**]	moves down one word
[1]-**[9],**[**A**]-[**F**]	the nybble value being edited
[**ENTER^**]:	inputs three zeroes as word value
[**<-**]:	first back-arrow prompts for a new address, second exits the editing
[**ON**]	turns the calculator OFF

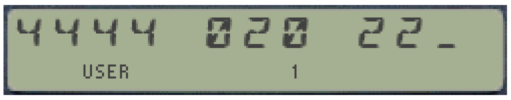
ROMED is taken from the DISASM ROM. Besides the changes *"of rigueur"* to use the Library#4 routines, I have made a couple of enhancements to the original implementation, adding the underscores for the edited field and the **[ENTER^]** control key for a closer resemblance with the HEPAX implementation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

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

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

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


ADR: 444_
USER 1

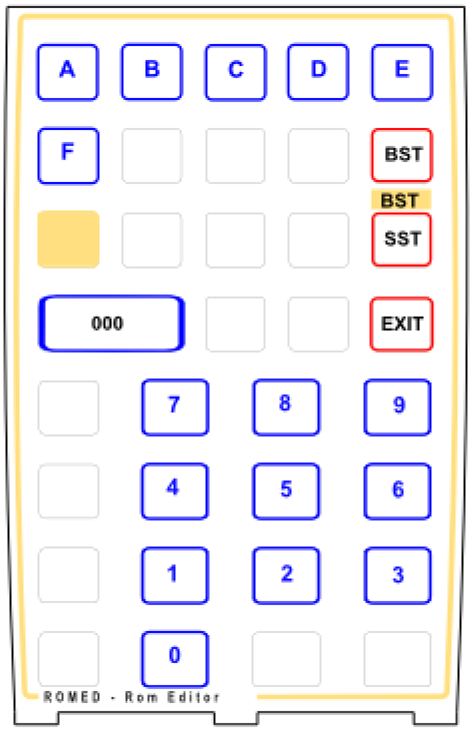

4444 020 22_
USER 1

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

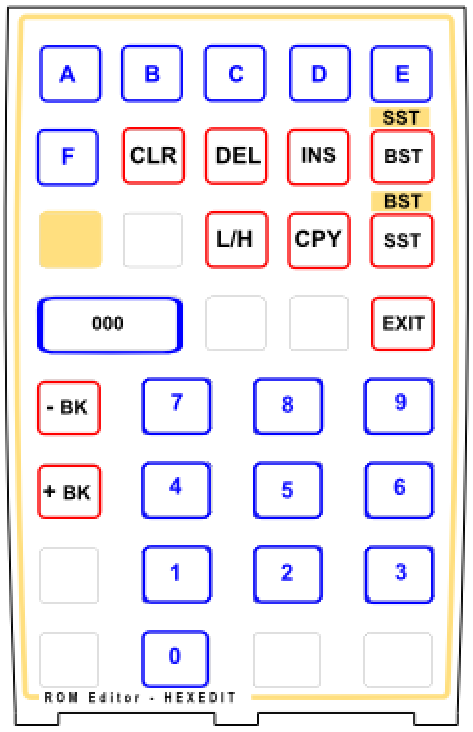
Final remark.- The original **ROMED** is available in the DISASM module, under the name **WROM,** and **RAMEDIT** is also included in the OS/X module, named **RAMED** there**.** Yes, a bit of a naming conflict but who said life was perfect?

A Quick Comparison.

The figure below shows a comparison between **ROMED** and the HEPAX HEX Editor **HEXEDIT –** with the latter literally a superset of the former one, so the same keyboard overlay could be used for both.



ROMED - Wrom



HEXEDIT - Hepax

5.2. ADVANCED MCODE FNS

5.2.1. First things first: Peeling the HEX Onion.

These functions address classic needs, subject of frequently asked and often misunderstood topics when it comes to the internals of the system and general Hex domain (I know I've been there myself for one, and more often than none!)

HEX>VSM	**Hex to VASM**	Converts prompt to VASM	Clifford Stern
VSM>HEX	**VASM to Hex**	Converts Address to HEX	Clifford Stern
BCDBIN	**From BCD-Hex to Binary**	Decimal input in X	Ken Emery
BINBCD	**From Binary to BCD-Hex**	NNN input in X	Ken Emery
GETW	**Get ROM Word**	Absolute adr in X (!)	Rafael Lorente
DISSST	**SST Disassembly**	Requires HEPAX	VM Electronics

- **BCDBIN** converts the decimal number in X into its binary representation, stored in the S&X field of the X-register as NNN output. The original argument is saved in LASTX. The valid range of values goes from 0 to 4,095 – thus covering all the S&X field from 0 to FFF.

- **BINBCD** is the inverse function, which decodes the S&X data in the NNN stored in X and returns its decimal equivalent to the X-register. The original NNN is saved in LASTX.

Note: If you're reading this you'll probably be aware of the OS routines [BCDBIN] (in the mainframe) and [BIN-BCD] (in the CX), so you have no doubt realized that the valid ranges accepted by the functions are larger than those for the OS counterparts, which only reach 256 = 0FF.

- **DISSST** is a token of appreciation for the HEPAX Module. It is the same program included in the HEPAX Manual, used for line-by-line to disassembling. You need to input the initial and final addresses, and press R/S for each consecutive MCODE line shown in the display. Naturally it requires the HEPAX Module plugged in the system, as the function **DISASM** does all the heavy lifting for you.

The FOCAL listing for **DISSST** is shown below:- easy does it when you have powerhouse functions behind!

```
01 LBL "DISSST
02 0
03 X<>F
04 "BEGIN: "
05 4
06 HPROMPT
07 "END: "
08 4
09 HPROMPT
10 CLA
11 4
12 DECODYX
13 RDN
14 LASTX
15 DECODYX
16 CODE
17 SIGN
18 SF 01
19 LBL 01
20 DISASM
21 STOP
22 SF 04
23 GTO 01
24 END
```

- **GETW** is an indulgence function – as a memento of the very early days of the author's MCODE experimentation many moons ago (when I should have been dedicating all my energies to passing my EE exams, but, well, I wasn't). It simple reads the ROM word located at the address specified in the X-register, as a decimal number. Valid ranges start at zero, and end at (16*4,096 –1) = 65,535. The word value is of course stored in the S&X field of register X (as FETCH would do and actually does within **GETW**).

  As an example, to read the first word of the ROM plugged in page "C" (which happens to be the POWER_CL in my system) , you'll type:

  12*4096= 49,152.0000, then **GETW**, **DCD** -> 00C = XROM id#12

- **HEX>VSM** and **VSM>HEX** are what you need when trying to read the HP VASM Listings – which are arranged in a Quad-based fashion, using Octal addresses within each page (from 0 to F) and quad (from 0 to 3). This, let's say it clearly, is a pain the rear back for anyone other than the original HP developers (and coming to think about that, maybe to them as well); so a conversion to/from Hex was in order – and thanks to Ken Emery we have it.

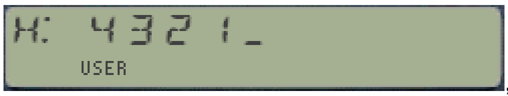
  These functions will prompt for the appropriate fields in each case, a beauty to behold and perhaps the only way to not make mistakes if you only use them sporadically. The prompts will look as follows:

1. **HEX>VSM** prompts for the Hex address; you need to input four digits and press R/S The result will be formatted according to the VASM convention. The number keys and the A through F keys are the only ones which are allowed for inputs. Once four digits have been entered, no more may be keyed in.

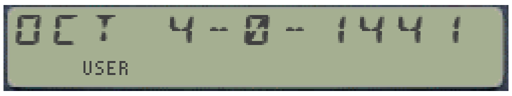
2. **VSM>HEX** prompts sequentially for the Page number, quad number, and Octal address (also four digits) separated by hyphens in the display. Press R/S to see the result, an address in Hex. The range of legal addresses is 0000 to 1777. Digits outside this range will not be accepted by the routine. If the address is less than 1000, you must key in a leading zero.

For example, let's convert the address 0x4321 in VASM form and back to "normal" :

XEQ "HEX>VSM", at the prompt "H:" type further: "4, 3, 2, 1"

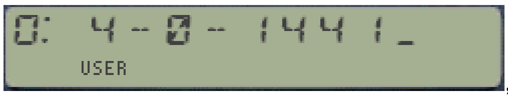

H: 4321_
USER

, R/S ->

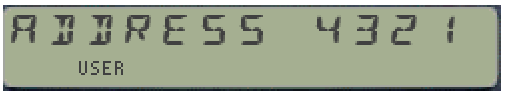

OCT 4-0-1441
USER

And now the reverse:

XEQ "VSM>HEX", at the prompt "O:" type the data fields:


O: 4-0-1441_
USER

, R/S ->


ADDRESS 4321
USER

5.2.2. Jumps and Executions: Hex codes from Mnemonics.

Welcome to the land of the bold and the brave M-Coder! The following sections deal with many important functions, really indispensable for the MCODE programmer.-

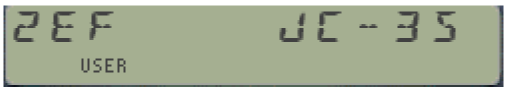
JC	**Jump if Carry**	Encodes Jump from X	D. Wilder / A. Martin
JNC	**Jump if Not Carry**	Encodes Jump from X	D. Wilder / A. Martin
CGO _ _ _ _	**GoTo if Carry**	Encodes GOTO code	W. Doug Wilder
CXQ _ _ _ _	**GoSub if Carry**	Encodes GOSUB code	W. Doug Wilder
NCGO _ _ _ _	**GoTo if Not Carry**	Encodes GOTO code	W. Doug Wilder
NCXQ _ _ _ _	**GoSub if Not Carry**	Encodes GOSUB code	W. Doug Wilder

What could be a better way to start than with the Jump and Execute aid functions? Written by Doug Wilder and available in different revisions of the DISASM ROM, this function set is an essential fixture in every system. I cannot imagine approaching MCODE writing without them, now you know one of my secret weapons!

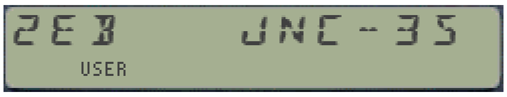
- **JC** and **JNC** encode the jump distance provided in the X-register (in decimal) into the appropriate hex code for a carry or no-carry jumps respectively. The argument can be positive or negative, depending of the jump direction. The value cannot be larger than 63 or 64 respectively, or the "NONEXISTENT" message will be shown.

  For example, to obtain the hex codes for jumps of 35 words back the current PC, we type:

  35, CHS, **JC** → for the carry set condition,


```
2EF        JC -35
   USER
```

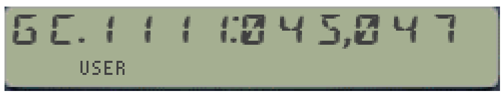
  35, CHS, **JNC**→ for the Not-carry condition.

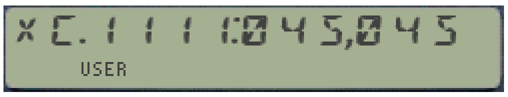

```
2EB       JNC -35
   USER
```

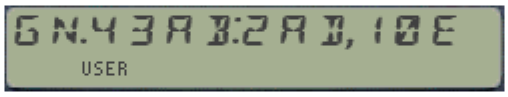
Notice how the result includes the mnemonic in the display, as a convenient feedback to remind you of the original requirement. You can take my word for that – this is a godsend functionality that spares you from repeat checking of jump tables, and the error-prone process associated to it.

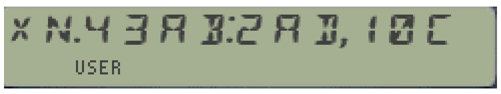
- **CGO**, **CXQ**, **NCGO**, and **NCXQ** all share the same user interface: a four-digit length prompt meant to input the GOTO or GOSUB addresses, anywhere in the I/O Bus of the HP-41 system (i.e. from 0000 to FFFF). Obviously this only makes sense if used with permanent addresses, like the OS, the Library#4, or some system-peripherals like the Time Module, printer or HP-IL Interface.

And who said the 12-chars LCD display was too small to be useful? Look at the concise yes absolutely clear representation of the information as returned to the display by any of these four functions; a beautiful compromise to say the least:


```
GC.1111:045,047
   USER
```


```
xC.1111:045,045
   USER
```


```
GN.43AB:2AB,10E
   USER
```


```
xN.43AB:2AB,10C
   USER
```

Even taking the risk of insulting your intelligence, allow me to clear out the meaning of the different fields in the display.

The three distinct areas include the type of jump, the hex address, and the hex codes used to program it. The table below explains the meaning of the type characters:

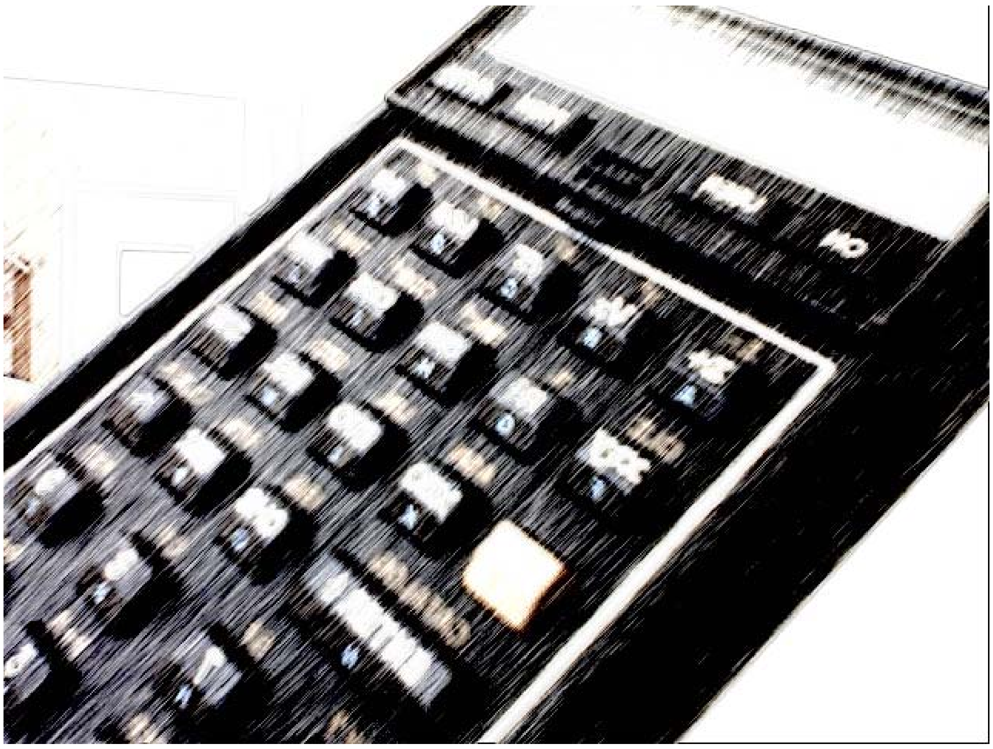
Symbol	Condition	Description
GC	?C GO	If Carry GOTO
XC	?C XQ	If Carry GOSUB
GN	?NC GO	If No-Carry GOTO
XN	?NC XQ	If No-Carry GOSUB

The other two fields in the display provide the address given in the prompt, and the two hex codes corresponding to the appropriate instructions. Therefore you always have confirmation that the input address was correct – quite a fundamental requirement and thus worth re-assuring in each result.

Summarizing, the four examples listed above corresponded to the following cases:

?CGO 1111	→	045,047	;	?CXQ 1111	→	045,045
?NCGO 43AB	→	2AD,10E	;	?NCXQ 43AB	→	2AD,10C

Now you'd understand how useful these functions have been when putting together all those Library#4 calls from the different Library#4-aware modules. Can you imagine doing it without these functions? I honestly can't!

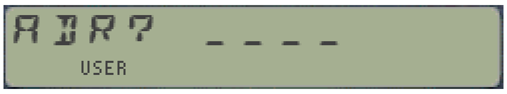


5.3. ADVANCED FOCAL FNS

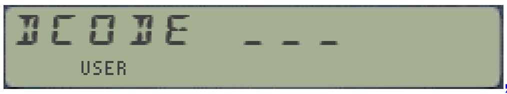
5.3.1. Last Treats and Tricks

ADR?	**Address Encoder**	Encoded NNN in X	W. Doug Wilder
DCODE _ _ _	**NNN to HEX string**	NNN in X, RG# in prompt	Frits Ferwerda
FDATA _	**Function Data**	Prompts for F. Name	
HEXIN _	**HEX Input**	1-9, and A-F	Håkan Thörngren
HEXNTRY	**HEX Keyboard**	1-9, and A-F	Clifford Stern
COMPILE	**Compiles jump distances**	Global LBL in ALPHA	Frits Ferwerda
NOPS	**Finds NOPs**	BBB|EEE as NNN in X	Frits Ferwerda
PC<>RTN	**Exchanges PC and RTN**	Get your head spiining!	W&W GmbH
XQ>XR	**XEQ to XROM**	Converts instructions	W&W GmbH

- **ADR?** provides an easy and convenient way to encode system addresses as NNN in the X-register. The outcome of ADR? Can be used by **RAMEDIT** directly, and by **RAMED** after you copy it into ALPHA (using CLA, ARCL X for instance). The prompt expects an Hex value with four digits, and the keyboard is redefined to only accept numeric keys and letters A-F as valid data options. Use the back-arrow key to cancel, as always.



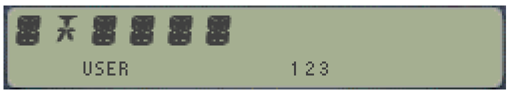
- Functions **DCODE** is the ubiquitous NNN->HEX function present in every ROM worth its name (ML ROM, HEPAX, TOOLBOX...). We can't have enough of a good thing, or so it seems... This implementation will prompt for the register absolute address in the prompt.

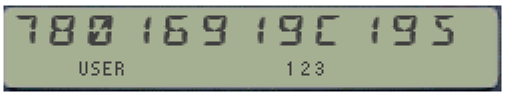


, wich allow arguments up to 999 dec

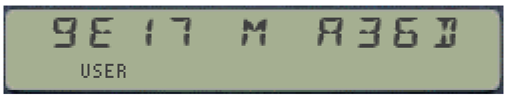
DCODE reads and decodes in ALPHA the contents of the register which absolute address is in X (in program mode) or given at the prompt (in RUN mode). No stack drop is performed. Register address is checked for existence. DCODE is equivalent to **VRG** in the OS/X Module, and can be thought as the combination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

As an example to impress your friends, decode the contents of the Status register 12("c") = Switch on ALPHA to see the complete contents scroll.



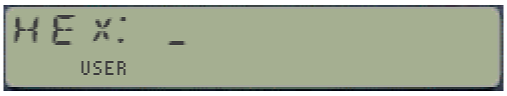
→


- **FDATA** returns the relevant data for a function in a plug-in module, including its FAT coordinates, and code location address. It also indicates whether it is an MCODE or a FOCAL program. You need to enter the function name at the prompt (note that <u>ALPHA will be switched on automatically</u> for you), or have it ready in ALPHA if used in a program. The result will be placed in ALPHA, and shown in the display if in RUN mode. For example, applying it to itself: XEQ "**FDATA**", then enter "F, D, A, T, A", ALPHA:



- **HEXNTRY** is the well-known function published in Ken Emery's book "*MCODE for beginners*", and originally written by Clifford Stern. The calculator keyboard is redefined to be a HEX-keypad, with the numeric keys and letters A to F available for data input. For all purposes it supercedes **CODE** (or **CDE**), which is available in the AMC_OS/X module anyway.

- **HEXIN** does basically the same thing, except that it uses the text in Alpha (if any) as prompt (quite useful in programs). Use Back-Arrow to delete digits and R/S to terminate the data entry.

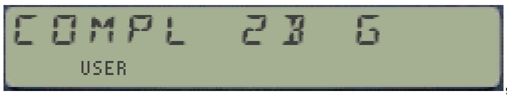
  

  HEX: _
  USER

  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

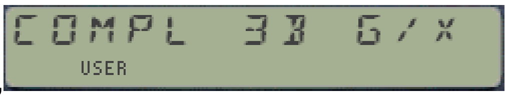
- **COMPILE** is a very powerful function that writes all the jump distances in the GOTO and XEQ instructions within the program named in ALPHA. This is extremely useful when uploading a program to a Q-RAM device, like the HEPAX RAM. Having all the jumping distances compiled expedites the execution of 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.

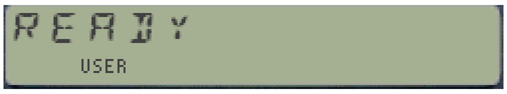
    COMPL 2B G
    USER

    ,

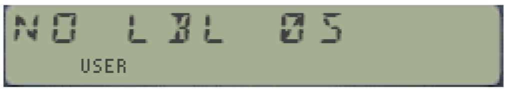
    

    COMPL 3B G/X
    USER

  - 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:

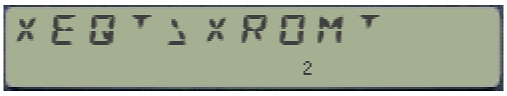
    READY
    USER

    NO LBL 05
    USER

- **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.

  XEQ⊤ \ XROM⊤
  2

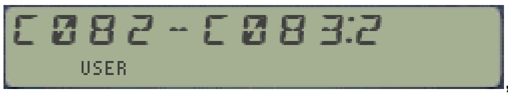
  will be shown while the conversion occurs.

  This function is taken from the RAMBOX OS, written by W&W GmbH (authors of the world-class, emblematic CCD Module).

- **NOPS** is a sleek function taken from the ML ROM, a great resource for the dedicated M-Coder. It searches the locations within the addresses specified in X as a NNN (in the form BBB|EEE) looking for two or more consecutive NOPS. If found, the locations and number of NOPs are reported in the display, and the NNN in X is updated so that it can be re-used by **NOPS** again to locate the next "hole" within the initial address range.

  Let's see an example. I have the POWER_CL module plugged in page 12 in my system, and I'm curious to see where the NOPS are. Say I'm too lazy to open the ROM blueprints so let's use **NOPS** instead. The first step is to provide the address range to scan, which we'll do using **HEXIN** as follows: **HEXIN**, "C, 0, 0, 0, C, F, F, F", R/S -> beautiful NNN in the X-reg.

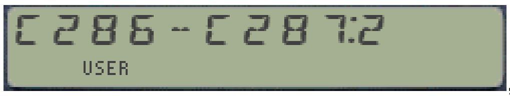
  Next we called **NOPS**, witch immediately reports the first group found (which corresponds to the FAT-END pair):



C082-C083:2
USER

,

  Since the NNN in X has been appropriately updated by the function, we bravely execute NOPS again, returning:



C286-C287:2
USER

,

  and repeating this until the end we can compile the following table:

Addresses	Number of NOPs	Cumulative SUM
C082 – C083	2	2
C286 – C287	2	4
C2A3 – C2C4	34	38
C5FA – C61B	34	72
C7C0 – C7CB	12	84
CA00 – CA01	2	86
CA85 – CA86	2	88
CC4E – CC56	9	97
CD96 – CD98	3	100
CDB1 – CDB3	3	103
CFCB – CFF2	40	143
CFF4 – CFF8	5	148

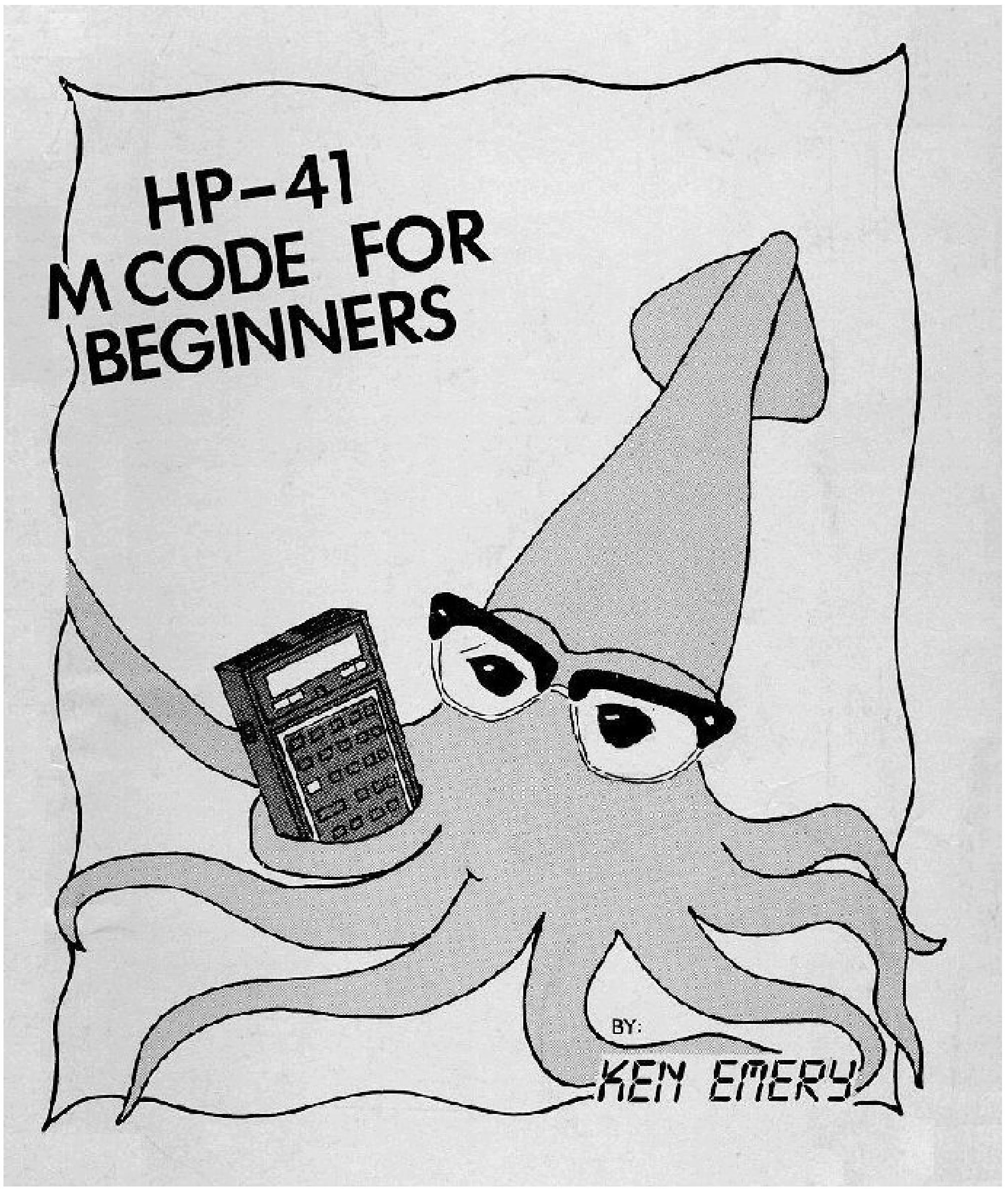
  When there's no more NOPS to be found the function returns 'NO"; an if used in a program it will skip the next program line (as FALSE condition).

  Note: As a way to verify all these intermediate results, we can use sub-function **PGROOM** within the POWER_CL Module to count the total number of NOPs – which should be equal or greater then the sum of all the partial results returned by the repeat executions of **NOPS** (which was 148 in this case). When we do that, we get:

  12, **XQ2** "PGROOM" → 97,000 Oooops !!

  Trouble in paradise??? – not so, what we're seeing is the typical effect of a bank-switched module: **PGROOM** is located in bank-3 of the Power_CL, thus it has found 97 NOPS in that bank, which is *not the same as those existing in the main bank*, or bank-1. If we looked at the Bank-1 blueprint we'd see that there are 234 NOPS in it, which fits the bill as it is larger than 148 indeed. In fact, the remaining (234 – 148) = 86 are single-NOPs scattered around the bank-1 code. Tricky but lovely :-)

That's all folks, this concludes the RAMPAGE & TOOLBOX 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.- VREG Program Listing.

As mentioned in page 4, **VREG** was removed from the RAMPAGE module to allow the inclusion of **FLCOPY**. It is but a very simple routine (albeit may become useful at times); see below the FOCAL listing in case you’re interested in having it available on your machine.

Input: bbb.eee in X
Output: sequential listing of Rnn with their contents

```
01  LBL "VREG"
02  CF 21
03  CF 29
04  LBL 00
05  FIX 0
06  "R"
07  ARCL X
08  "|-: "
09  FIX 4
10  ARCL IND X
11  AVIEW
12  PSE
13  ISG X
14  GTO 00
15  SF 29
16  END
```

All in all, not much to write home about, thus hopefully you agree **FLCOPY** is a much more interesting function to have in the toolset instead.

The irony here is that such a simple FOCAL program is more code-efficient than an equivalent MCODE implementation of the same functionality; so you see sometimes FOCAL has its point.

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 accessory parameters – like the pointers in 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.- Extended 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 <u>linking register</u> of PREVIOUS module (or zero if first block)
NXT:	address of <u>top register</u> of NEXT module (or zero if last block)
TOP:	address of <u>top register</u> 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

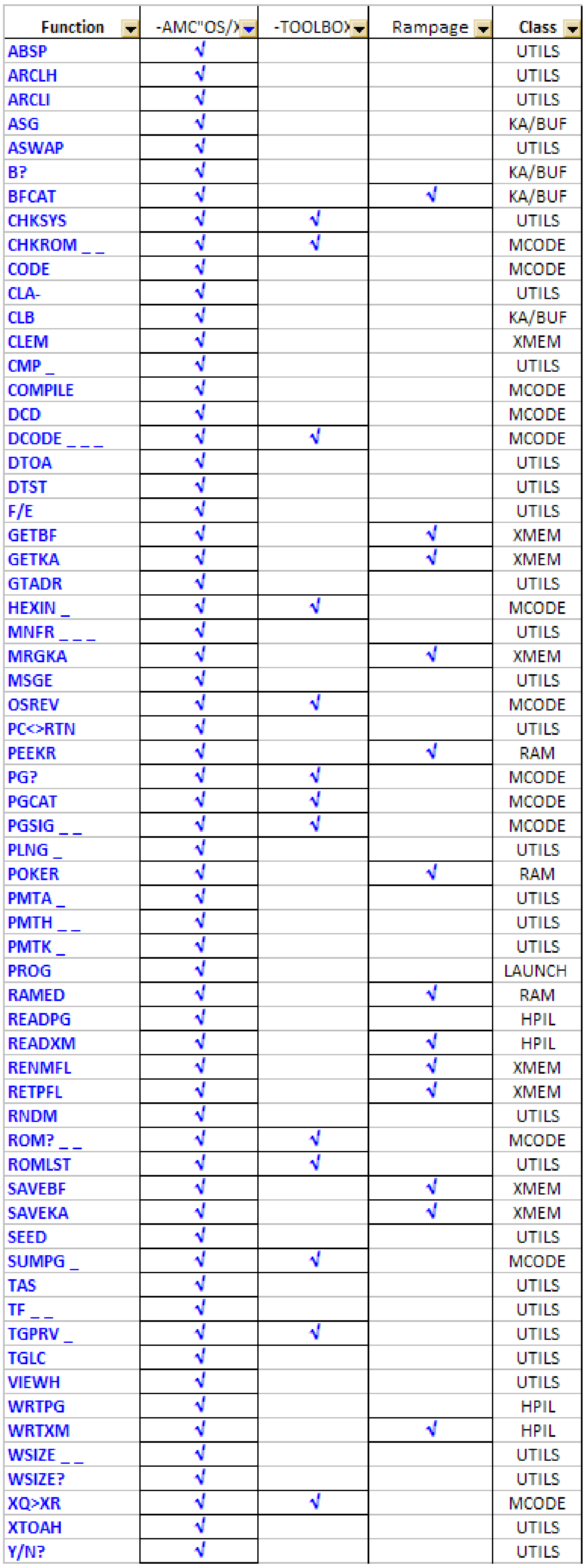
Appendix 4.- Function Repeats.

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.

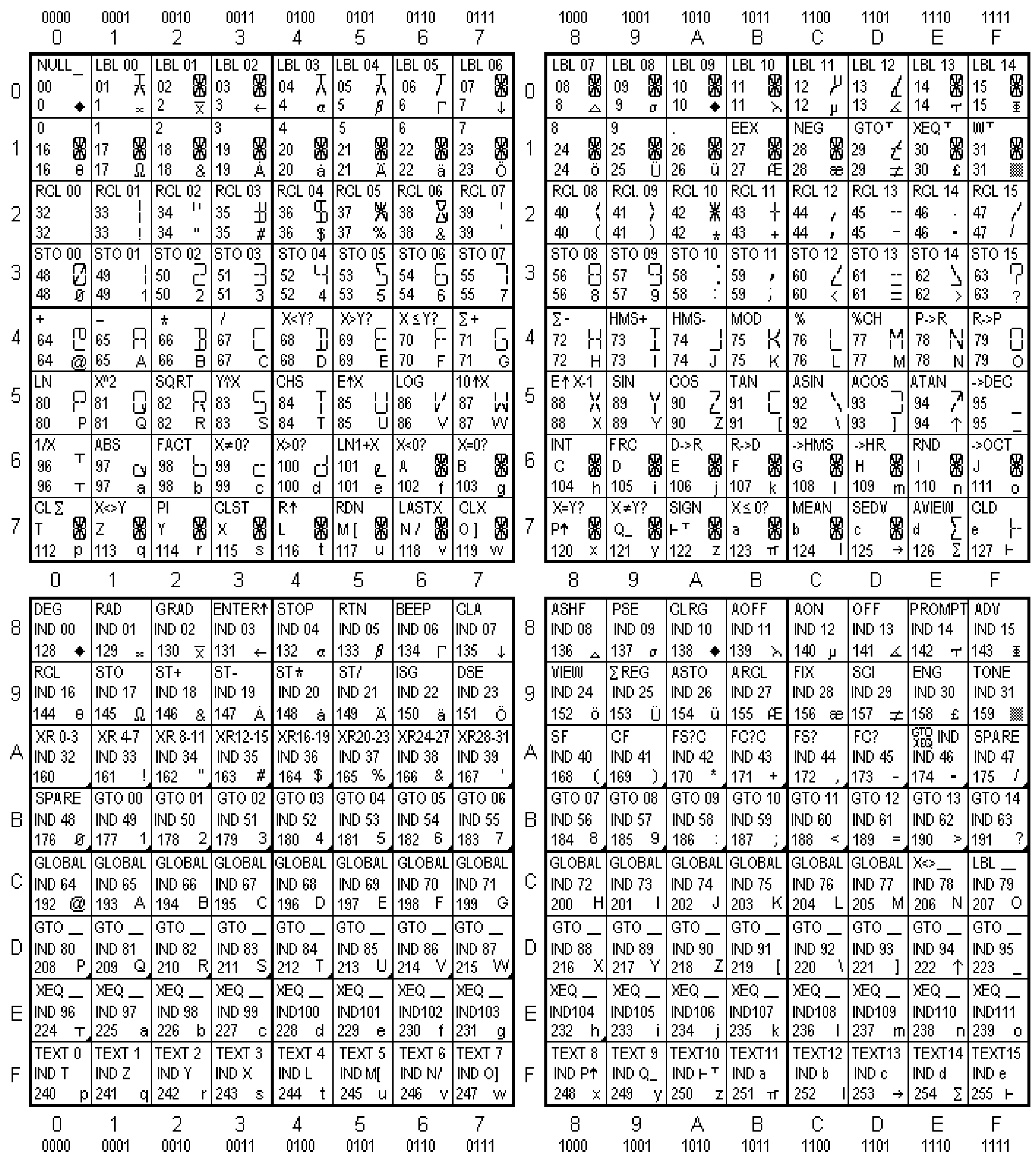
The stats are as follows:

38 unique functions,
12 dup fns in the TOOLBox,
14 dup fns in the RAMPAGE.



Function	-AMC"OS/	-TOOLBO	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 0.- HP-41 Byte Table



	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7
0	NULL_ 00 0 ◆	LBL 00 01 1 ∝	LBL 01 02 2 x̄	LBL 02 03 3 ←	LBL 03 04 4 α	LBL 04 05 5 β	LBL 05 06 6 Γ	LBL 06 07 7 ↓
1	0 16 16 θ	1 17 17 Ω	2 18 18 &	3 19 19 Å	4 20 20 å	5 21 21 Ä	6 22 22 ä	7 23 23 Ö
2	RCL 00 32 32	RCL 01 33 33 !	RCL 02 34 34 "	RCL 03 35 35 #	RCL 04 36 36 $	RCL 05 37 37 %	RCL 06 38 38 &	RCL 07 39 39 '
3	STO 00 48 48 Ø	STO 01 49 49 1	STO 02 50 50 2	STO 03 51 51 3	STO 04 52 52 4	STO 05 53 53 5	STO 06 54 54 6	STO 07 55 55 7
4	+ 64 64 @	– 65 65 A	* 66 66 B	/ 67 67 C	X<Y? 68 68 D	X>Y? 69 69 E	X≤Y? 70 70 F	Σ+ 71 71 G
5	LN 80 80 P	X^2 81 81 Q	SQRT 82 82 R	Y↑X 83 83 S	CHS 84 84 T	E↑X 85 85 U	LOG 86 86 V	10↑X 87 87 W
6	1/X 96 96 ⊤	ABS 97 97 a	FACT 98 98 b	X≠0? 99 99 c	X>0? 100 100 d	LN1+X 101 101 e	X<0? A 102 f	X=0? B 103 g
7	CL Σ T 112 p	X<>Y Z 113 q	PI Y 114 r	CLST X 115 s	R↑ L 116 t	RDN M [117 u	LASTX N / 118 v	CLX O] 119 w
8	DEG IND 00 128 ◆	RAD IND 01 129 ∝	GRAD IND 02 130 x̄	ENTER↑ IND 03 131 ←	STOP IND 04 132 α	RTN IND 05 133 β	BEEP IND 06 134 Γ	CLA IND 07 135 ↓
9	RCL IND 16 144 θ	STO IND 17 145 Ω	ST+ IND 18 146 &	ST- IND 19 147 Å	ST* IND 20 148 å	ST/ IND 21 149 Ä	ISG IND 22 150 ä	DSE IND 23 151 Ö
A	XR 0-3 IND 32 160	XR 4-7 IND 33 161 !	XR 8-11 IND 34 162 "	XR12-15 IND 35 163 #	XR16-19 IND 36 164 $	XR20-23 IND 37 165 %	XR24-27 IND 38 166 &	XR28-31 IND 39 167 '
B	SPARE IND 48 176 Ø	GTO 00 IND 49 177 1	GTO 01 IND 50 178 2	GTO 02 IND 51 179 3	GTO 03 IND 52 180 4	GTO 04 IND 53 181 5	GTO 05 IND 54 182 6	GTO 06 IND 55 183 7
C	GLOBAL IND 64 192 @	GLOBAL IND 65 193 A	GLOBAL IND 66 194 B	GLOBAL IND 67 195 C	GLOBAL IND 68 196 D	GLOBAL IND 69 197 E	GLOBAL IND 70 198 F	GLOBAL IND 71 199 G
D	GTO __ IND 80 208 P	GTO __ IND 81 209 Q	GTO __ IND 82 210 R	GTO __ IND 83 211 S	GTO __ IND 84 212 T	GTO __ IND 85 213 U	GTO __ IND 86 214 V	GTO __ IND 87 215 W
E	XEQ __ IND 96 224 ⊤	XEQ __ IND 97 225 a	XEQ __ IND 98 226 b	XEQ __ IND 99 227 c	XEQ __ IND100 228 d	XEQ __ IND101 229 e	XEQ __ IND102 230 f	XEQ __ IND103 231 g
F	TEXT 0 IND T 240 p	TEXT 1 IND Z 241 q	TEXT 2 IND Y 242 r	TEXT 3 IND X 243 s	TEXT 4 IND L 244 t	TEXT 5 IND M[245 u	TEXT 6 IND N/ 246 v	TEXT 7 IND O] 247 w

	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0	LBL 07 08 8 △	LBL 08 09 9 σ	LBL 09 10 10 ◆	LBL 10 11 11 λ	LBL 11 12 12 μ	LBL 12 13 13 ∡	LBL 13 14 14 ⊤	LBL 14 15 15 Ξ
1	8 24 24 ö	9 25 25 Ü	. 26 26 ü	EEX 27 27 Æ	NEG 28 28 æ	GTO T 29 29 ≠	XEQ T 30 30 £	W T 31 31 ▒
2	RCL 08 40 40 (	RCL 09 41 41)	RCL 10 42 42 *	RCL 11 43 43 +	RCL 12 44 44 ,	RCL 13 45 45 –	RCL 14 46 46 .	RCL 15 47 47 /
3	STO 08 56 56 8	STO 09 57 57 9	STO 10 58 58 :	STO 11 59 59 ;	STO 12 60 60 <	STO 13 61 61 =	STO 14 62 62 >	STO 15 63 63 ?
4	Σ- 72 72 H	HMS+ 73 73 I	HMS- 74 74 J	MOD 75 75 K	% 76 76 L	%CH 77 77 M	P->R 78 78 N	R->P 79 79 O
5	E↑X-1 88 88 X	SIN 89 89 Y	COS 90 90 Z	TAN 91 91 [	ASIN 92 92 \	ACOS 93 93]	ATAN 94 94 ↑	->DEC 95 95 _
6	INT C 104 h	FRC D 105 i	D->R E 106 j	R->D F 107 k	->HMS G 108 l	->HR H 109 m	RND I 110 n	->OCT J 111 o
7	X=Y? P↑ 120 x	X≠Y? Q_ 121 y	SIGN ⊢T 122 z	X≤0? a 123 π	MEAN b 124 |	SEDV c 125 →	AVIEW d Σ 126 Σ	CLD e ⊦ 127 ⊢
8	ASHF IND 08 136 △	PSE IND 09 137 σ	CLRG IND 10 138 ◆	AOFF IND 11 139 λ	AON IND 12 140 μ	OFF IND 13 141 ∡	PROMPT IND 14 142 ⊤	ADV IND 15 143 Ξ
9	VIEW IND 24 152 ö	ΣREG IND 25 153 Ü	ASTO IND 26 154 ü	ARCL IND 27 155 Æ	FIX IND 28 156 æ	SCI IND 29 157 ≠	ENG IND 30 158 £	TONE IND 31 159 ▒
A	SF IND 40 168 (	CF IND 41 169)	FS?C IND 42 170 *	FC?C IND 43 171 +	FS? IND 44 172 ,	FC? IND 45 173 -	GTO XEQ IND IND 46 174 ▪	SPARE IND 47 175 /
B	GTO 07 IND 56 184 8	GTO 08 IND 57 185 9	GTO 09 IND 58 186 :	GTO 10 IND 59 187 ;	GTO 11 IND 60 188 <	GTO 12 IND 61 189 =	GTO 13 IND 62 190 >	GTO 14 IND 63 191 ?
C	GLOBAL IND 72 200 H	GLOBAL IND 73 201 I	GLOBAL IND 74 202 J	GLOBAL IND 75 203 K	GLOBAL IND 76 204 L	GLOBAL IND 77 205 M	X<>__ IND 78 206 N	LBL __ IND 79 207 O
D	GTO __ IND 88 216 X	GTO __ IND 89 217 Y	GTO __ IND 90 218 Z	GTO __ IND 91 219 [	GTO __ IND 92 220 \	GTO __ IND 93 221]	GTO __ IND 94 222 ↑	GTO __ IND 95 223 _
E	XEQ __ IND104 232 h	XEQ __ IND105 233 i	XEQ __ IND106 234 j	XEQ __ IND107 235 k	XEQ __ IND108 236 l	XEQ __ IND109 237 m	XEQ __ IND110 238 n	XEQ __ IND111 239 o
F	TEXT 8 IND P↑ 248 x	TEXT 9 IND Q_ 249 y	TEXT10 IND ⊢T 250 z	TEXT11 IND a 251 π	TEXT12 IND b 252 |	TEXT13 IND c 253 →	TEXT14 IND d 254 Σ	TEXT15 IND e 255 ⊢

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.