HP-41 Module
"XROM" ROM

Overview

The XROM Module contains a collection of MCODE function utilities with the usability and convenience themes in mind. Some are taken from relatively obscure modules, such as the Proto-CODER and the NFCROM – and others are from bigger collections or less-known modules, in an attempt to increase their usage and to provide a more portable vehicle for all users.

It comes without saying that thanks and credit should go to the original authors as listed in the function table below, with the new functions shown in white background. For the most part the other functions are modified only slightly to take advantage of Library#4 routines – which is therefore required for this module. You're nevertheless encouraged to read the original module manuals for additional insights.

This module is designed to be used independently from others – except the Library#4. There's very little redundancy with the RAMPAGE or TOOLBOX modules, and therefore can also be used as an extension to them.

XROM	Function	Description	Author	Source
31,00	-XROM ROM	Section Header	*Ángel Martin*	This project
31,01	**AFCN? _**	Function Data	*Poul Kaarup*	PK Collection
31,02	**ASRCH _**	Fuction Data	*Klaus Huppertz*	Prisma Magazine 1/90
31,03	**CPUF**	CPU Frequency	*Doug Wilder*	BLDROM
31,04	**JUMP1 _**	Mcode J1 Codes	*Poul Kaarup*	PK Collection
31,05	**JUMP3 _**	Mcode J3 Codes	*Poul Kaarup*	PK Collection
31,06	**HEX2ROM _**	Hex to Xrom	*Greg McClure*	GJM ROM
31,07	**HEX2RM+ _**	ditto - appended	*Greg McClure*	GJM ROM
31,08	**OSREV**	OS revision	*Nelson F. Crowle*	NFC ROM
31,09	**PCAT _**	Port Catalog	*Mark Power*	Debugger ROM
31,10	**PROMT _**	Custom Hex Prompt	*Nelson F. Crowle*	NFC ROM
31,11	**ROM2HEX _**	Xrom to Hex	*Greg McClure*	GJM ROM
31,12	**ROM2HX+ _**	Ditto - appended	*Greg McClure*	GJM ROM
31,13	**SPLASH**	Splash Screen	*Nelson F. Crowle*	This project
31,14	**XROM _ _:_ _**	Call any Xrom function	*Clifford Stern*	Mcode for Beginners
31,15	**XROM$ _**	Lists XROM calls in program	*Klaus Huppertz*	Prisma Magazine 4/90
31,16	**?LIB4**	Lib#4 Existence Test	*Ángel Martin*	This project
31,17	-}-}-}-VV-}-}	Section Header	*Ángel Martin*	This project
31,18	**DEBUG**	Debug	*Clifford Stern*	Mcode for Beginners
31,19	**LOOP**	Loop	*Clifford Stern*	Mcode for Beginners

31,20	**RSLCT**	Ram selection	*Clifford Stern*	Mcode for Beginners
31,21	-XROM RAM	Section Header	*Ángel Martin*	This project
31,22	**BJUMP _ _**	Byte Jumper	*Nelson F. Crowle*	NFC ROM
31,23	**BYTE _ _ _**	Enters Byte	*Klaus Huppertz*	Prisma Mag 2-3/91
31,24	**CDOWN _ _**	Curtain Down	*Greg McClure*	GJM ROM
31,25	**CUP _ _**	Curtain Up	*Greg McClure*	GJM ROM
31,26	**CURT?**	Finds Curtain location	*Poul Kaarup*	PK Collection
31,27	**CURTAIN _ _ _**	Sets curtain	*Poul Kaarup*	PK Collection
31,28	**GETST _ _**	Get Status XM File	*Ángel Martin*	This project
31,29	**INSBYT#**	Insert Byte	*Fritz Ferwerda*	ML ROM
31,30	**KAFLP**	KA Flip (all keys)	*Ángel Martin*	This project
31,31	**KYFLP _**	Key Flip	*Ángel Martin*	This project
31,32	**KYOFF**	Suspend Key	*Fritz Ferwerda*	ML ROM
31,33	**LB _ _ _**	Load Byte	*Fritz Ferwerda*	ML ROM
13,34	**LOADB _ _**	RAM Byte Editor	*Nelson F. Crowle*	PCODER_1A
31,35	**LODB _ _**	Load Byte(s)	*Nelson F. Crowle*	NFC ROM
31,36	**PC<>RTN**	Exchanges PC and RTN-1	*W&W GmbH*	CCD Module
31,37	**PRBYTES _ _**	Print Buffer Bytes	*HP Co.*	HP-IL Devel
31,38	**POPADR**	Pops RTN addr	*Håkan Thörgren*	RAMPage Module
31,39	**POPRTN**	Pops complete RTN Stack	*Poul Kaarup*	PK Collection
31,40	**PUSHRTN**	Pushes complete RTN stack	*Poul Kaarup*	PK Collection
31,41	**RTN?**	any address in stack?	*Doug Wilder*	BLDROM
31,42	**RTNS**	RTN Stack levels	*Ángel Martin*	This project
31,43	**RCLBYT#**	Recall byte	*Fritz Ferwerda*	ML ROM
31,44	**STOBYT#**	Store Byte	*Fritz Ferwerda*	ML ROM
31,45	**SAVEST _ _**	Saves Status in XM File	*Ángel Martin*	This project
31,46	**STVIEW**	Stack View	*Ángel Martin*	This project
31,47	**XMXEQ _**	Calls program in XM	*Klaus Huppertz*	Prisma Mag 4/89 p14
31,48	**XRCL _ _**	Extended-Reg Recall	*Ángel Martin*	This project
31,49	**XSTO _ _**	Extended-Reg Store	*Ángel Martin*	This project
31,50	**XX<> _ _**	Extended-Reg Exchange	*Ángel Martin*	This project

Splash Screen. *(by Nelson F. Crowle)*

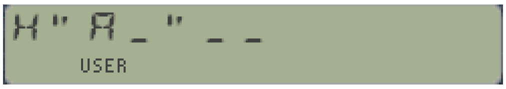
The ultimate display demo that unfortunately does not work on V41 but will beautifully show on real machines (41-CL included). Watch the letters moving across the LCD window to form the welcome message "#4 ON-LINE" . Seeing is believing!

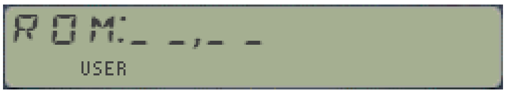
XROM to and from HEX bytes. *(by Greg McClure)*

Sometimes it is needed to translate between XROM indents (##,##) and the FOCAL bytes that represent the XROM function (Ax, xx). Function **HEX2ROM** prompts H"A_"_ _ and expects three hex digits (of which the first can't be > 7). On successful entry of the 3rd hex digit the corresponding XROM value will be placed into the Alpha register and displayed in the form: "XROM_ _ , _ _" .

Function **ROM2HEX** does the reverse. It prompts ROM: _ _ , _ _ and expects 4 decimal values (with max for the first pair is 31, and max for the second pair is 63). On successful entry of the 4th decimal digit the corresponding hex bytes will be placed in the Alpha register and displayed in the form: "HEX'_ _:_ _"

If at any time during entry for any of these function the opposite function is desired, pressing the "H" key will switch to the opposite routine (**ROM2HEX**<>**HEX2ROM**).



<-->


Appended to ALPHA versions

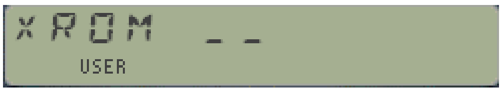
Function **HEX2RM+** prompts H"A_"_ _ and expects three hex digits (of which the first can't be > 7). On successful entry of the 3rd hex digit the corresponding XROM value will be appended to the Alpha register and displayed in the form: "XROM_ _ , _ _" .

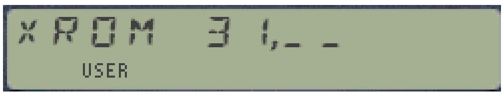
Function **ROM2HX+** does the reverse. It prompts ROM: _ _ , _ _ and expects 4 decimal values (max for the first pair is 31, max for the second pair is 63). On successful entry of the 4th decimal digit the corresponding hex bytes will be appended to the Alpha register and displayed in the form: "HEX'_ _:_ _"

If at any time during entry for any of these function the opposite function is desired, pressing the "H" key will toggle between opposite routines (**ROM2HX+**<>**HEX2RM+**).

Calling XROM Functions *(Clifford Stern)*

This is one of my favorite-ever functions: use it to call a function from any plug-in module by entering the function id# numbers at the prompts. A beautiful example of superb MCODE programming that uses the OS routines to its best.





Subroutine RTN Stack Functions.

This groups deals with the RTN Stack. The OS has provision for up to six levels of subroutines; that is your FOCAL programs can have up to five chained XEQ calls to other programs or subroutines.

The program pointer (PC) and the first two pending return addresses are stored in status registers b(12), the third is stored as two halves on each register, and the remaining three in status register a(11).

b(12):

R	3	A	D	R	2	A	D	R	1	P	C	N	T
13	12	11	10	9	8	7	6	5	4	3	2	1	0

a(11):

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

Getting Information on Subroutine Levels usage.

- **RTN?** is a test function that checks whether any return level exists. The result is YES/NO depending on the case, and in a program execution the following line will be skipped it false.
- **RTNS** returns the number of pending RTN levels to the X register. Obviously the result will be zero if executed in manual mode, as no pending subroutines exist. The stack is lifted.
- **POPADR** removes one pending routine address off the RTN stack and shifts the rest one level down. No output to X is produced (so it's more like XQ>GO despite its name).
- **PC<>RTN** exchanges the program counter and the first RTN address. In a running program this causes the execution to jump back to the pending address and then return to the calling point, i.e. it is effectively another way to execute the subroutine twice.

Extending the Subroutine Levels capacity. *(by Poul Kaarup)*

You can use these functions to push and pop the entire RTN stack into a buffer (id#7) in the I/O area.

- **PUSHRTN** saves the contents of the RTN stack in the buffer and resets it to zero for an extended RTN stack with 6 more levels capacity.
- **POPRTN** overwrites the current (extended) RTN stack with the buffer contents saved previously (i.e. the original RTN stack).

These functions are obviously meant to be used as a pair. Note also that the buffer#7 will be erased when you switch the calculator OFF.

Saving Status Registers in X-Memory.

You can use functions **SAVEST _ _** and **GETST _ _** to make backup copies of the status registers into X-Memory files, and to restore their contents back to the status area. The functions prompt for the number of status registers to include in those back-up files, which must be at least one and not more than 16.

For example if you just want to save the stack registers T,Z,Y,X, and L then you'd enter "5" in the prompt (since the count always starts with register T as the first one). The file name is expected to be in ALPHA - thus register M (and possibly N) would be partially used by the function itself.

These functions are programmable. In a running program the number of status registers is taken from the program line after the function – which won't be entered into the X register but as the prompt value instead.

The Status files have a dedicated file type in X-Memory. If you're using the AMC_OS/X Module, then their entries will be marked with the 'T" prefix during the enumeration:

Playing with Key Assignments.

This module adds a couple of brand-new KA-related routines that you may find interesting. Their mission is to flip the key assignments on a given key or for the complete keyboard – so that the shifted and un-shifted assignments are mutually toggled.

- **KAFLP** toggles all key assignments – turning shifted ones into non-shifted, and vice-versa. This will only leave unassigned keys unchanged, but will reverse the assignments if only one assignment exists for the keys.

- **KYFLP_** prompts for a key to perform the same task on an individual key basis. The prompt includes the back-arrow key but will ignore the toggle keys (ON/USER & PRGM/ALPHA)

- **KYOFF _** prompts for a key to temporarily suspend its user assignments. You can restore them using **LKAON** from the AMC_OS/X or RAMPage modules.

In case you wonder why bother with this functionality, having the ability to toggle a key's USER key assignments becomes very handy if you have two function launchers assigned to that key. A good example is with the SandMath, SandMatrix and 41Z modules – the three of them "competing" for prime time on the [Σ+] key. Flipping the assignments will save you a lot of [SHIFT] key pressings to access the functions within those launchers.

A touch of PRISMA Utilities. *(by Klaus Huppertz)*

These four functions are interesting utility examples taken from PRISMA, the German User's club magazine. They were contributed by Klaus Huppertz.

ASRCH_ returns information on the function or FOCAL program which name is entered at the prompt (or in ALPHA during a running program). The information returned is quite complete, including the HEX number, the address and the type (User Code or, MCode). This version will also look for PROGRAM files in X-Memory if no function OR focal program (in main memory) exists with that name. –.For example, executing **ASRCH** on itself it returns: "9BFD M A7:C2"

BYTE _ _ _ is a byte-loading function, a very popular subject in the old days. It will let you enter the byte with value in the prompt at the current program counter location (PC), no more no less. Be careful where you perform the insertion as it may create havoc if you break the Label chain. More about this subject in next section of the manual.

XROM$_ prompts for a FOCAL program and scans the program looking for all the XROM calls included in it, showing either the section header for the module if it's plugged in, or the XROM function id# if the module is not present. The listing is sequential, and you need to press R/S to see the next match. It's therefore very handy to find out the XROM dependencies of your FOCAL code. Note however that it will not work on FOCAL programs loaded in plug-in ROMs.

XMXEQ _is an X-Mem Program File caller – but one that allows the program file to be in any of the X-Mem modules – the only limitation is that the code cannot cross the voids between modules. Should such a contingency occur, the function will warn you with a "broken goose" message:

The "NONEXISTENT" message will be shown if the file is not found or is not a PROGRAM file

Notice for all ALPHA prompting functions. - in manual mode the ALPHA prompt is offered automatically, no need to press the ALPHA key to start typing it. In a running program the function/program name is expected to be in ALPHA.

Curtain functions. *(by Greg McClure)*

The absolute register number that marks the location of R00 is often called the "curtain". Its value is kept in one of the system stack registers (c to be specific), as most synthetic programmers know. The system moves this value up or down depending on how many registers are specified with the SIZE instruction. A trick used by some synthetic programmers is to raise or lower the value of the "curtain" in a program.

If the value is raised by n, then R0 thru Rn-1 are hidden, and Rn becomes R0, Rn+1 becomes R1, etc. If the value is subsequently lowered by n (which must be done before exiting the program for reasons explained next) then these hidden registers are recovered and the original register numbers are restored.

Actually, when the "curtain" is raised in this way, the n registers affected temporarily become program steps as far as the O/S is concerned. Since this could confuse the system when doing a CAT 1, "curtain" raising and lowering should be used carefully. In addition, if a PACK occurs while the "curtain" is raised like this, the hidden registers could easily (and probably will) change values. If the "curtain" is raised to temporarily save registers, it should be lowered back before doing these system functions (CAT 1 or PACK, or similar functions).

Conversely, if the "curtain" is lowered and not raised back to its original value, certain labels and ENDs could be modified by simple RCL, STO and X<> instructions. This messes up the chain of CAT 1 and can lead to MEMORY LOST. However, used properly, "curtain" manipulation can be of great use to a programmer that needs to call a subroutine that uses the same registers as another program.

- Function **CURT?** returns to the X-register the absolute address (in decimal) corresponding to the current location of the curtain.
- Function **CURTAIN _ _ _** sets the curtain to the absolute address entered either in the three-digit prompt if used in manual mode, or contained in the X register when used in a program; but always as a decimal number.
- Function **CUP** _ _ raises the curtain up. The function will prompt for the number of registers to hide. If this function is entered in a program, the number of registers to hide should be entered as a value after the **CUP** function. This will be interpreted as the argument of **CUP**, not as a value to enter into X. It must be followed by a non-numeric entry function or the CPU will get confused.
- Function **CDOWN _ _** lowers the curtain down. The function will prompt for the number of registers to restore. If this function is entered in a program, the number of registers to restore should be entered as a value after the **CDOWN** function. This will be interpreted as the argument of **CDOWN**, not as a value to enter into X. It must be followed by a non-numeric entry function or the CPU will get confused.

Loading Bytes. *(N. F. Crowle and others)*

If you lived through the days of byte jumpers and load bytes' challenges, you'll no doubt have fond memories of what it was like to work with synthetics and PPC ROM routines. Here's a few MCODE functions that should rekindle your appreciation for those chores, now from an MCODE perspective.

A few functions come from the NFC ROM and the ProtoCoder_1A. Consider them modern day versions of some of those vintage routines, with a usability and convenience twist added by the MCODE implementation; as well as speed.

BJUMP __ is a byte jumper utility that prompts for the number of bytes to jump over, counted from the current PC. If you see no use for this function you can go ahead and ignore it altogether, but it sure would have been nice to have it back then…

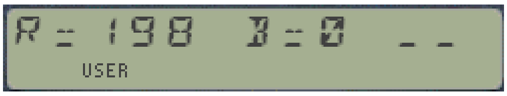
LB _ _ _ is an MCODE version of the Load Bytes routines. Enter the byte value in decimal format at its prompt, from 000 to 255 – to have the corresponding byte written in RAM at the current PC location. See the byte table in appendix-1 for details.

LODB _ _((_ _), _ _) is a very ingenious approach to solve the multi-byte loader problem. This function will prompt additional fields depending on the previous inputs – to complete the sequence required for 2-byte and three-byte instructions. The inputs are expected in Hexadecimal format, from 00 up to FF. See byte table for details.

Example: to enter ΣREG IND 25 you can use the prompt values "99" and "99" at the initial and subsequent prompts (see the display below at the point of the last digit input):

This function is not finished – but it works for the majority of 2- and 3-byte combinations as is. It was later superseded by a more systematic RAM-editor version in the Proto-Coder ROM, which is described below, but I think it still has a place for this module.

LOADB _ _ is a more capable RAM Editor that can be used to review and edit the contents at the byte level. It takes the starting position from the current PC location, and presents a prompt that shows the current register and byte number, as well as the byte value:



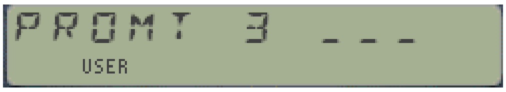
At this point you can use the [SHIFT]/[SST]keys to move up and down in memory, the [ENTER^] key to null the byte at that location, the [RCL] key to input a new RAM address, the [R/S] key to terminate the function and return to the OS, or the back-arrow key to edit the byte with a new value. Be careful with the changes you make, and be aware that pressing back arrow will require editing the byte value (i.e. no cancel from it).

INSBYT#, **STOBYT#**, and **RCLBYT#** are taken from the ML ROM and use a direct approach to inserting, storing and recalling bytes in RAM. They expect the RAM location to be in the "BRRR" format (b: byte number within the register, from 0-6; RRR: absolute register address in hex).

The RAM location needs to be formatted in the two Least-Significant Bytes of the X register (or Y-register for **INSBYT#** and **STOBYT#**) as a binary number – which can be accomplished using any direct hex entry function, such as PROMT described below.

Function	INSBYT#	SAVEBYT#	RCLBYT#
X Input	Byte value in decimal	Bute value in decimal	BRRR as binary
Y Input	BRRR as binary	BRRR as binary	n/a

PROMT _ is a general-purpose, direct HEX entry function. The number of Hex digits to enter is provided at its own prompt in manual mode, or in the X register if used in a program. The result is placed in X as a binary number – with as many valid digits as the number of fields in the prompt.



For example, to prepare the RAM location "60FF" using **PROMT**, you first enter "4" at the function prompt and then the four hex digits of the address directly. The result is placed in the X register ready for the byte functions to use.

Note: This function is very similar to **HPROMPT,** included both in the HEPAX and the Hepax_Dis-Assembler Modules.

Function Information. *(by Poul Kaarup)*

AFCN?_ Displays information about a function in several sections or lines. R/S brings up the next lines of information. Back-arrow aborts the function. The function name is to be entered at the ALPHA prompt.

For native HP41 functions (like DEC):

- ADR= 132B the functions address i.e. OS ROM
- DEC= 004,095 the decimal value input to get DEC
- HEX= 04,5F the hexadecimal value to get DEC
- MAINFRAME indicates a native HP41 function

For plug-in modules (like AFCN?):

- ADR= B295 the functions address ie ROM
- DEC= 161,078 the decimal value input to get AFCN?
- HEX= A1,4E the hexadecimal value to get AFCN?
- XROM 05,14 XROM# for AFCN?

MCODE Jumps. *(by Poul Kaarup)*

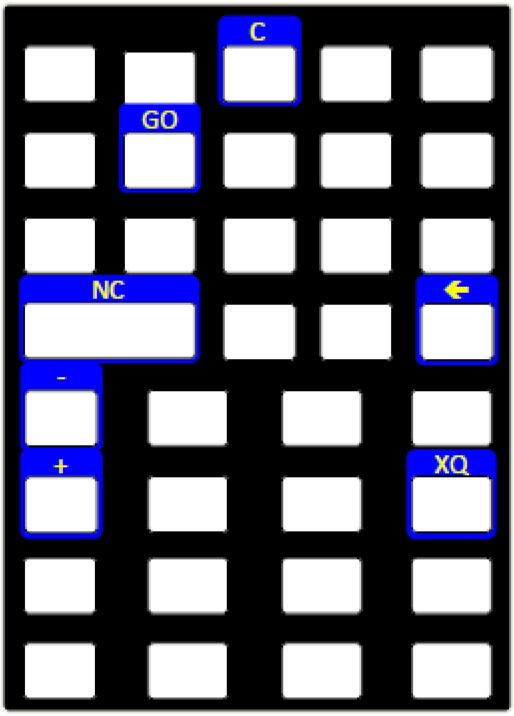
These two functions are taken from the PK_Collection. With them you can find out the hex codes corresponding to the jumping distances for type-3 jumps, or for jump-to locations for type-1 jumps.

Simply type the function name **JUMP1** or **JUMP3** to start building the prompts. The functions are clever enough to only let you input the allowed characters, as follows:

- **JUMP1_** shows a question mark "?" and waits for your input. At this point it expects either "C" or "N" for the Carry- or not-Carry flavors, then you follow suit by entering either "G" or "X' for the GO or XEQ version of the jump.

- **JUMP3_** shows "J" and waits for your input – which can only be either "C" or "N" for the carry- and not-carry flavors; then you follow suit with "-" or "+" to indicate the jumping direction, and finally the distance value in bytes in HEX (!).

These functions are not programmable. The overlay below summarizes all the options for them as described above. In all cases the back-arrow key will take you back to the previous stage of the prompt, or terminate the function if already at the "root" level.



Note that a set of equivalent functions is available in the TOOLBOX module, as follows:

	JUMP1		JUMP3	
	?NC GO _ _ _ _		**JC**	Value in X
	?NC XQ _ _ _ _		**JNC**	Value in X
	?C GO _ _ _ _			
	?C XQ _ _ _ _			

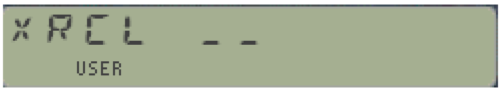
Extended Registers Storage.

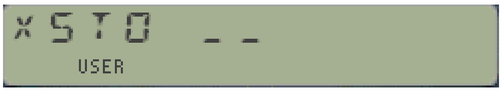
If you've ever run out of data registers and wished there was a "back-door" mechanism to use in emergencies, then you should find this section interesting. These functions operate on a I/O buffer located below the .END. and above the Key assignment area.

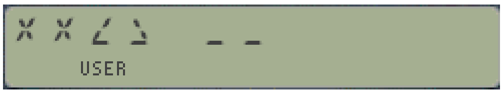
The buffer holds five extra registers for standard data storage, labeled XR-01 to XR-05 (therefore there's no XR-00 to speak of). Just enter the index for the extended register in the prompt and the data will be stored, recalled, or exchanged with the stack X-register – as if they were standard data registers.

- **XRCL _ _** recalls to the X register the content of the extended reg. which index is provided in the prompt, or in the next program line if used in a running program.
- **XSTO _ _** stores the X-register in the extended reg. given in the prompt, or in the next program line if used in a running program.
- **XX<> _ _** exchanges the contents of the X-register and the extended reg. which index is provided in the prompt, or in the next program line if used in a running program.

It you enter a non-valid index number (basically anything except 1,2,3,4,5) the prompt will be maintained - without an error condition – until you either cancel the function or enter a valid value. In program mode this will show a NONEXISTENT message and the execution will halt.



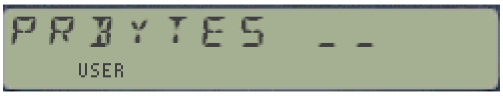




The buffer will be created the first time you save data in the extended registers, or attempt to retrieve it from them. The buffer registers XR-01 and XR-02 are shared by the RTN stack PUSH/POP functions, so be careful not to override their content if both features need to be used together. This buffer is not automatically created by the XROM module so the data will not survive a power-on/off cycle.

Other Functions. *(Diverse authors)*

PRBYTES Prints the bytes in an I/O buffer in hexadecimal format. The bytes are printed starting from the buffer pointer and terminating and the end of the buffer. If the HP-82143A printer is plugged in, it'll be used for printing. If not, the HP-IL printer will be used only if flag 33 is clear and there is no other controller on the loop. The mode switch on either printer must be set to TRACE or NORM, or flag 15 or flag 16 must be set for HP-IL printers other than the HP 82162A. if neither printer is present, the bytes will be displayed at about two bytes per second. Pressing [R/S] will exit the function. Pressing any other key will slow the display rate to about one byte per second.



Note that this version of **PRBYTES** is a prompting function. It will accept as input in its prompt any buffer id# on the I/O area, and not only the IL-Devel buffer type (with id# 12). If other buffer type is used, the pointer will be taken as 0 (top of the buffer) and the enumeration will only work properly assuming the buffer size is a multiple of seven.

PCAT Accesses a CATalog enumeration starting at the page provided in the function prompt. For example, press "5" to start with the TIME Module, or "7" to commence at page-7. The catalog will only show the header functions (those which names start with a hyphen), so it's up to you to stop it and press ENTER^ to see the functions included under each section.

CPUF returns a suitable surrogate for the CPU frequency. This function is a curious gem, although I'm not completely sure I managed to transcribe it well. It's supposed to return the number of CPU cycles per second, so I thought it'd be ideal for the CL given the different TURBO modes. Alas, it always returns the same value (1,126.316), irrespective of the TURBO setting. This is about 6 times bigger than the normal HP-41 result, (167,333) for what is worth. We have Doug Wilder to thank (again) for writing it, using the Time module to keep pace with things.

OSREV shows the revisions of the three OS ROMS in pages 0-2, for example for an unmodified 41-CX it returns:

Debugging functions. *(Clifford Stern)*

These three functions are taken from Ken Emery's book "MCODE for Beginners". Refer to that source for additional information on usage and limitations. Also you should be aware that Mark Power's DEBUGGER Module is a vastly superior approach to this purposes!!

- **DEBUG** Inserts a break-point in an MCODE program and halts execution at that point, allowing you to see the CPU registers and pointers
- **LOOP** Allows you to debug a loop within an MCODE program
- **RSLCT** Allows you to see the RAMSLCT pointer and the T register

ABS reg	nybbles													
#	13	12	11	10	09	08	07	06	05	04	03	02	01	00
487	0	0	RTN 3				RTN 2				RTN 1			
488	KY		RTN 4				XY		P	Q	G		ST	
489	CPU register C													
490	CPU register A													
491	CPU register B													
492	CPU register M													
493	CPU register N													
494	STATUS register T													
495	STATUS register Z													
496	STATUS register Y													
497	STATUS register X													
498	STATUS register L													
499	STATUS register M													
500	STATUS register N													
501	STATUS register O													
502	STATUS register P													
503	STATUS register Q													
504	STATUS register ⊢													
505	STATUS register a													
506	STATUS register b													
507	STATUS register c													
508	STATUS register d													
509	STATUS register e													
510								BREAK address				word		
511								BREAK address+1				word		
#	13	12	11	10	09	80	07	06	05	04	03	02	01	00

XY bit	07	06	05	04	03	02	01	00
CPU flag #	13	12	11	10	9	8	v	w

	v	w
0 =	hex mode	SLCT P
1 =	dec mode	SLCT Q

Appendix 0.- HP-41 Byte Table

	0 (0000)	1 (0001)	2 (0010)	3 (0011)	4 (0100)	5 (0101)	6 (0110)	7 (0111)
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

	8 (1000)	9 (1001)	A (1010)	B (1011)	C (1100)	D (1101)	E (1110)	F (1111)
0	LBL 07 08 8 △	LBL 08 09 9 σ	LBL 09 10 10 ◆	LBL 10 11 11 ⋋	LBL 11 12 12 μ	LBL 12 13 13 ∡	LBL 13 14 14 ┬	LBL 14 15 15 ≡
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 126 Σ	CLD e 127 ⊢
8	ASHF IND 08 136 △	PSE IND 09 137 σ	CLRG IND 10 138 ◆	AOFF IND 11 139 ⋋	AON IND 12 140 μ	OFF IND 13 141 ∡	PROMPT IND 14 142 ┬	ADV IND 15 143 ≡
9	VIEW IND 24 152 ö	Σ REG 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.