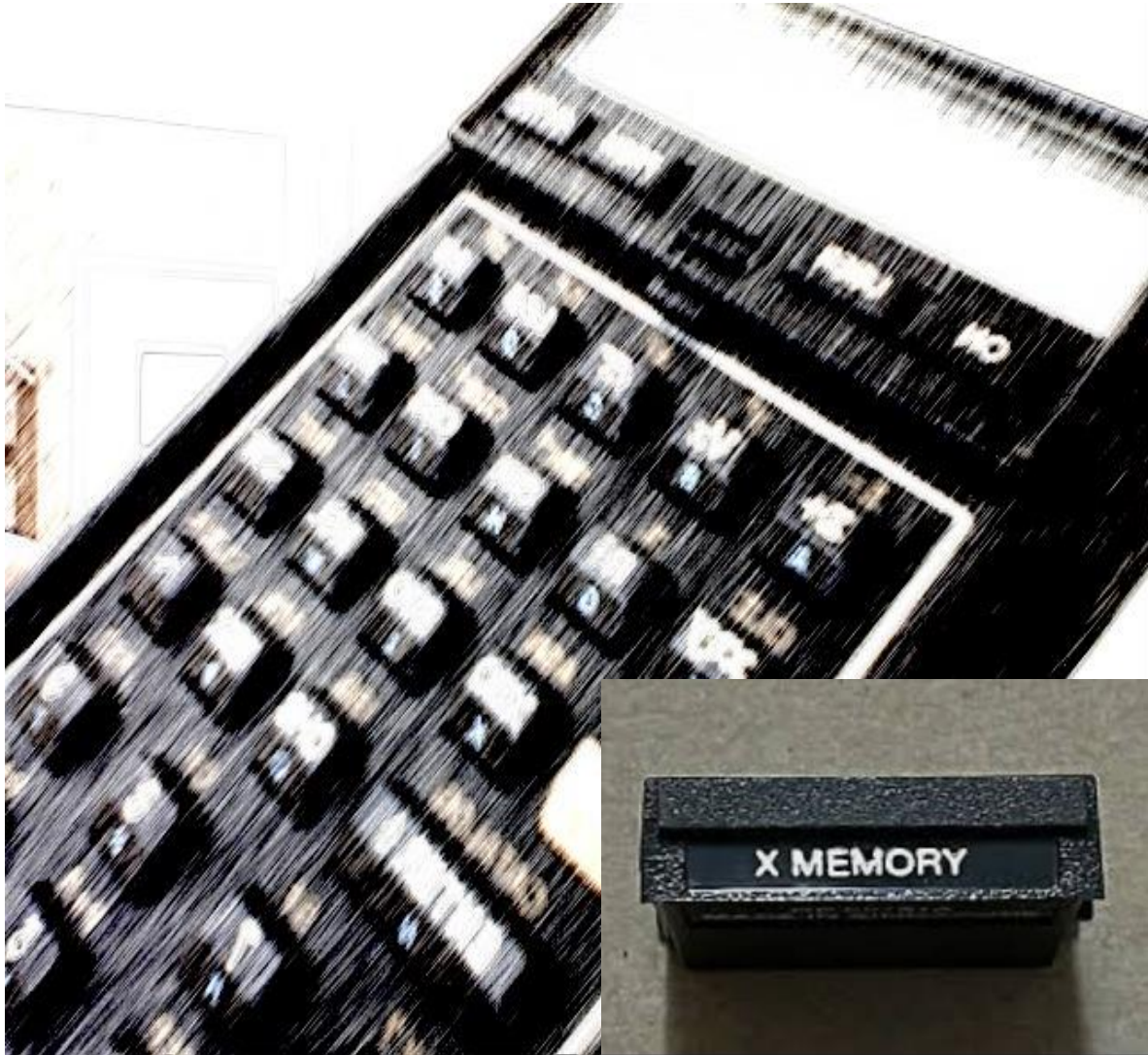


X-MEM X-FUNCTIONS HP-41 Module

Tools & Utilities for X-Memory

User's Manual and QRG.



*Written and programmed by Ángel M. Martin
June 10, 2022*

This compilation revision 1.1.2

Copyright © 2022 Ángel Martín

Published under the *GNU software licence agreement*.

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

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow.

See www.hp41.org

Acknowledgments.- This manual and the described module would obviously not exist without the wonderful functions and routines included in it. 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.

X - MEM X - F N C S M o d u l e X-Mem Tools & Utilities for the HP-41CX

1. Introduction.

This module holds a collection of functions dedicated to enhancing and expanding the Extended Memory handling. Some of them address missing aspects of functionality not covered in the X-Functions Modules, but others expand the potential of X-Memory into new areas, such as the LIFO Utilities and the In-File record management functions.

The module is 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 to make other general arrangements.

This manual is structured around the four sections in the module, as follows:

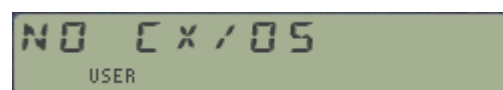
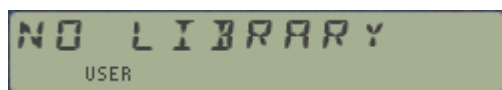
- XMEM XFNS Includes general-purpose RAM utils, RAM Editors and X-Mem extensions.
- DFL RECS In-File record management of the individual registers of a data File.
- LIFO UTILS Using X-Mem as a LIFO buffer to hold system date
- HP-IL XFNS Advanced functions for HP-IL

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

The module uses the Library#4 – but in a not bank-switched configuration. Using the Library#4 allowed for a substantial increase in the number and kind of functions compared with the standard approach.

The last remark is regarding the CX dependency: the module is 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.



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

Function index at a glance.-

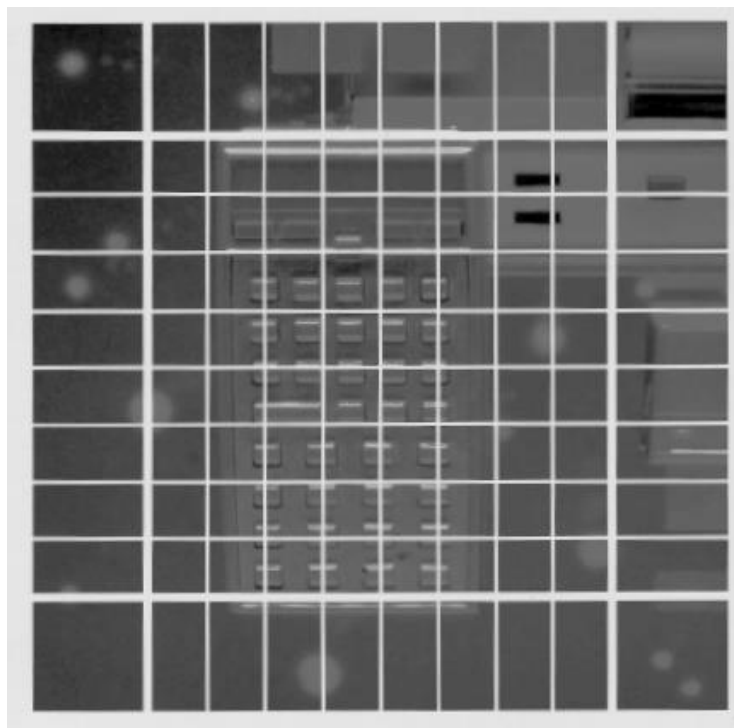
Here you have it, a decent collection of X-Memory functions that will no doubt enhance your HP-41 experience and open the door for new applications in your own programs.

XROM#	Function Name	Description	Input	Author
17,00	-XMEM XFNS	<i>Section Header</i>	<i>Shows splash Lib#4 splash</i>	<i>Nelson F. Crowle</i>
17,01	A<>RG _ _	Swaps Alpha and Regs.	Prompts for RG#	Ángel Martin
17,02	A<>ST	Swaps Alpha and Stack	None	Ángel Martin
17,03	ADVREC	Advances File Pointer	Pt. Increment in X	Ángel Martin
14.,04	ARCLCHR	ARCL Char	File Name in Alpha	Håkan Thörngren
17,05	ARCLIP _ _	ARCL Integer Part	Decimal number in X	Frits Ferwerda
17,06	CLMM	Clear Main Memory	needs OK in Alpha	Zengrange
17,07	CLRAM	Clears RAM	needs OK in Alpha	Ray del Tondo
17,08	CLXM	Clear Extended Memory	needs OK in Alpha	Zengrange
17,09	FLCOPY	Copy File (like-to-like)	"Source, Destination" in Alpha	Ángel Martin
17,10	FLHD	File Header	File Name in Alpha	Ángel Martin
17,11	FLTYPE	File Type	File Name in Alpha	Ángel Martin
17,12	GETBF	Restores Buffer from EM	File Name in Alpha	Håkan Thörngren
17,13	GETKA	Get Keys	File Name in Alpha	Håkan Thörngren
17,14	GETST	Get Status Rgs	File Name in Alpha	Ángel Martin
17,15	MRGKA	Merge Keys	File Name in Alpha	Håkan Thörngren
17,16	RAMED _	RAM Editor	Address in M or PRGM pointer	Zengrange
17,17	RAMEDIT _	RAM Editor	Address in X or PRGM pointer	Håkan Thörngren
17,18	REC-	Backs pointer by one	None	Ángel Martin
17,19	REC+	Advances pointer by one	None	Ángel Martin
17,20	RENMLFL	Rename File	Old Name, New Name in Alpha	Ángel Martin
17,21	RETPFL	Re-type File	Old, New types in X	Ángel Martin
17,22	RSTCHK	Reset Checksum	Program File Name in Alpha	Håkan Thörngren
17,23	SAVEBUF	Saves Buffer in EM	Id# in X, File Name in Alpha	Håkan Thörngren
17,24	SAVEKA	Save Keys	File Name in Alpha	Håkan Thörngren
17,25	SAVEST	Save Status Registers	File Name in Alpha	Ángel Martin
17,26	SORTFL	Sort Data File	Data File Name in Alpha	CCD Module
17,27	ST<>RG _ _	Swaps Stack and RG	Prompts for RG#	Ángel Martin
17,28	WORKFL	Get Work File	none	Sebastian Toelg
17,29	XQXM	Execute XM Program File	Program File Name in Alpha	Ross Wentworth
17,30	-DFL REGS	<i>Section Header</i>	<i>None</i>	<i>n/a</i>
17,31	"DFED	Data File Editor	Shows & Edits Records	Ángel Martin
17,32	D>H	Decimal to Hex	Dec value in X	Derek Amos
17,33	FLRCL _ _	Recall File Record to X	Rec# in prompt	Ángel Martin
17,34	FLSTO _ _	Stores X in file Record	Rec# in prompt	Ángel Martin
17,35	FLVEW _ _	View File Record	Rec# in prompt	Ángel Martin
17,36	FX<> _ _	Swaps X and File Record	Rec# in prompt	Ángel Martin
17,37	H>D	Hex to Decimal	Hex value in Alpha	Derek Amos
17,38	PEEKR	NNN Recall	Absolute address in X	Ken Emery
17,39	POKER	NNN Store	Absolute address in X	Nelson F. Crowle
17,40	-LIFO UTILS	<i>Section Header</i>	<i>n/a</i>	<i>n/a</i>
17,41	LIFOINI	Initializes INI buffer	Data File Name in ALPHA	Doug Wilder
17,42	ΣPUSH	PUSH Launcher	Prompts for function	Ángel Martin
17,43	^A	Saves Alpha to file	LIFO File Name in ALPHA	Doug Wilder
17,44	^F	Saves Flags to file	LIFO File Name in ALPHA	Doug Wilder
17,45	^RTN	Saves RTN in two recors	LIFO File Name in ALPHA	Doug Wilder
17,46	^ST	Saves Stack in 4 records	LIFO File Name in ALPHA	Doug Wilder
17,47	^X	Saves X in file record	LIFO File Name in ALPHA	Doug Wilder
17,48	^Z	Saves X,Y Regs	LIFO File Name in ALPHA	Doug Wilder

XROM#	Function Name	Description	Input	Author
17,49	ΣPOP	POP Launcher	Prompts for function	Ángel Martin
17,50	POPA	Restores ALPHA	LIFO File Name in ALPHA	Doug Wilder
17,51	POPF	Restores System Flags	LIFO File Name in ALPHA	Doug Wilder
17,52	POPRTN	Restores RTN Stack	LIFO File Name in ALPHA	Doug Wilder
17,53	POPST	Restores XYZT Stack	LIFO File Name in ALPHA	Doug Wilder
17,54	POPX	Restores X Register	LIFO File Name in ALPHA	Doug Wilder
17,55	POPZ	Restores X,Y Registers	LIFO File Name in ALPHA	Doug Wilder
17,56	-HPIL XFNS	<i>Section Header</i>	<i>n/a</i>	<i>n/a</i>
17,57	FLENG?	Drive File Length	File Name In ALPHA	Unknown
17,58	READF	Reads File from Drive	File Name in ALPHA	R. del Tondo
17,59	READPG	Reads 4k-Page from drive	File Name in ALPHA	Skwid
17,60	READXM	Reads EM from MassStg	File Name in Alpha	Skwid
17,61	WRTDF	Write Data File to Drive	File Name in ALPHA	R. del Tondo
17,62	WRTPG	Writes Page to Drive	Page# in X	Skwid
17,63	WRTXM	Write EM to MassStg	File Name in Alpha	Skwid

A few functions like CLRAM and the memory editors RAMEDIT and RAMED go beyond X-Memory per se, as they can be used to manually edit any area within the calculator's memory. The same can be said about general-purpose utilities like PEEKR and POKER, etc.

Some functions have the same name as equivalent ones in other modules, like the AMC_OS/X and the RAMPAGE & TOOLBOX. This duplication is somehow inevitable to have self-contained modules, so there's no need to have them plugged simultaneously.



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

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

- **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 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 in a program simply use **ST<>RG** typing any numbers for the prompt. No second line is needed in this case because the first data register is zero. To swap the Alpha {M,N,O,P} registers and data registers R05 to R08 you need two program lines, as follows:

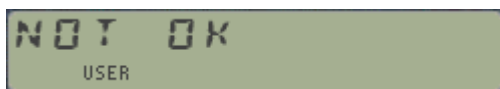
```
nn      A<>RG
nn+1    5
```

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.

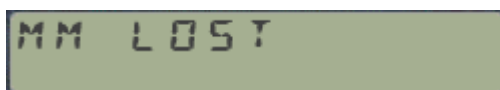
1.1.2. Clearing Memory – selectively or wholesale

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

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:

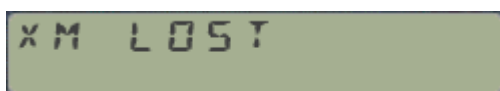


- **CLMM** erases the calculator Main memory, including Stack & Data Registers, Programs, and I/O Area - Key assignments and buffers (Alarms included). It will however leave X-Memory untouched. Note that **CLMM** stores nulls into every register, and in addition all status registers and flags are restored 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.



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 control 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).



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.



is generated when executed in RUN mode

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.

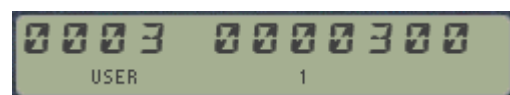
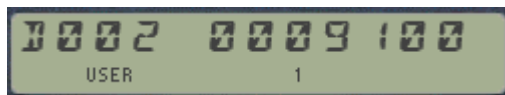
The starting register address is taken from the X register in RUN mode either as a decimal value between 0 and 999, or an a NNN with the address in the rightmost two bytes. The latter form allows for a direct entry to a byte value within the register. In PROGRAM mode it uses the current program pointer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

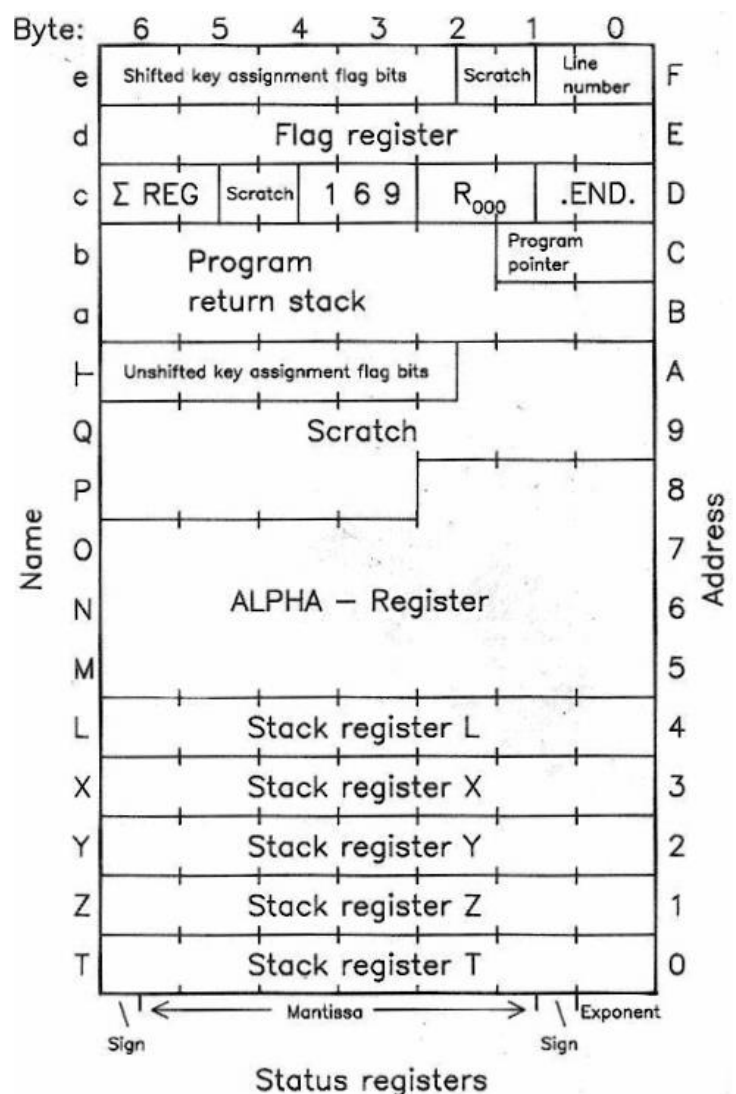
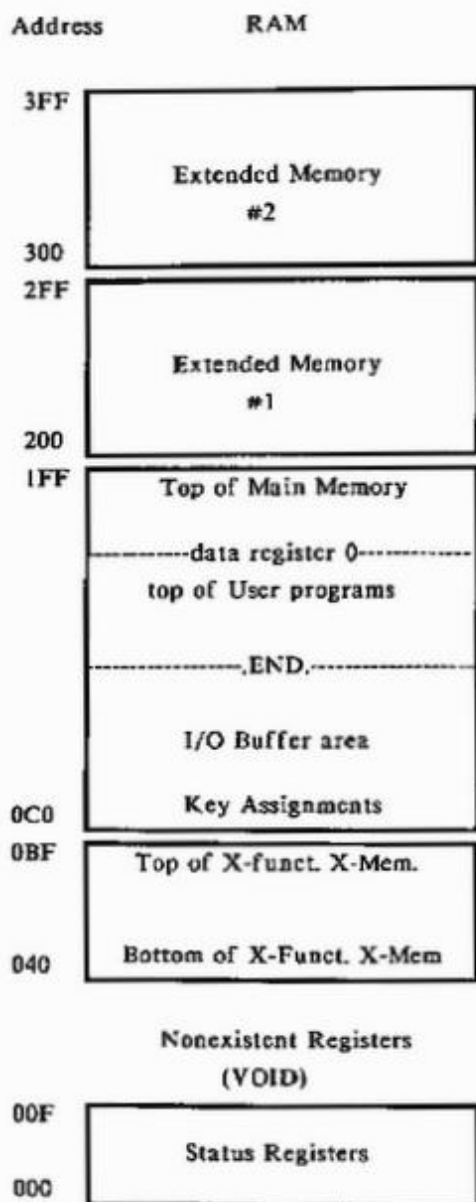


The control keys for RAMEDIT are as follows:

- [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.

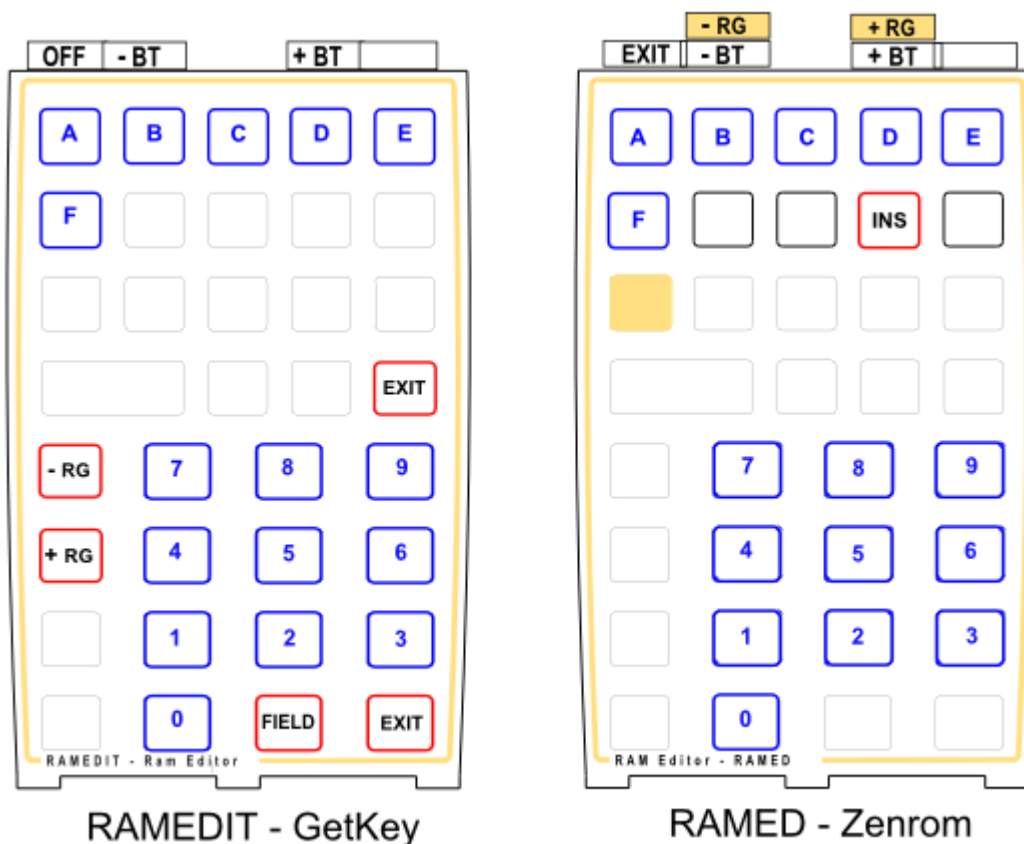
RAMED also redefines the HP-41 keyboard during execution to allow forward or backwards movements through memory in byte or register increments by pressing the [**USER**] and [**PRGM**] keys, (for bytes) and their **shifted** version for registers.

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.



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.

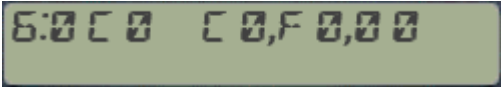
As an example, let's take a look at the key assignments registers, which have a format as follows:

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)

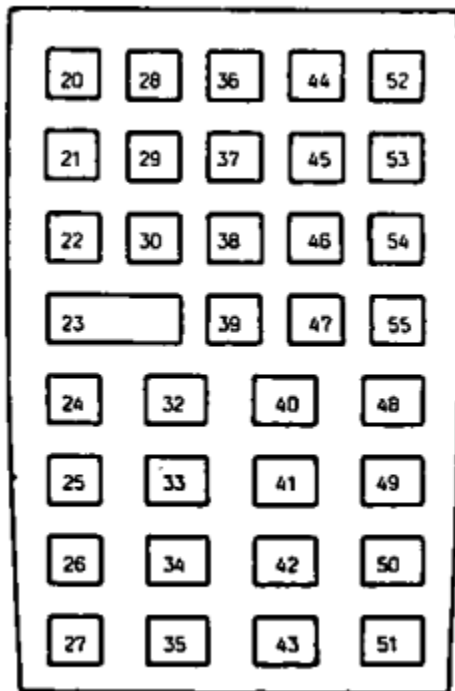
Suppose you wish to edit the lowest key assignment register, which is at address 0C0, and you want to go in at byte 6 of that register (that should contain F0). In standard RAMED notation this is address "6:0C0" – where the ":" character separates the byte from the register address.

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:

The image shows a digital display with a greenish-grey background. It displays the text "6:0C0" followed by a space and then "C0,F0,00". The characters are in a monospaced font, typical of a calculator display.

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.-

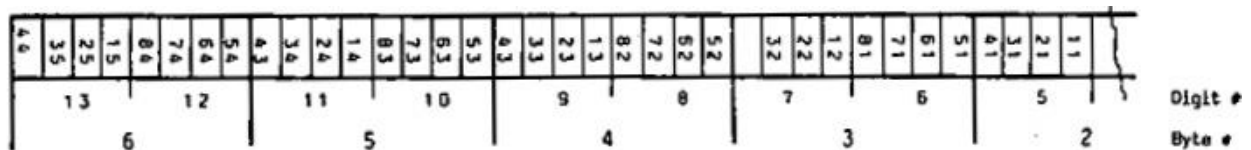


Basically the trick consists of setting the appropriate bits in status register 10 ("|-") and 15 ("e"), depending on whether it's a un-shifted or shifted assignment.

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	<i>Håkan Thörngren</i>
FLHD	File Header	Returns address to X	<i>Ángel Martin</i>
FLCOPY	File Copy	'Source, Destination' in ALPHA	<i>Ángel Martin</i>
FLTYPE	File Type	Gets file type to X	<i>Ángel Martin</i>
GETBF	Get Buffer from file	Buf id# in X, File Name in Alpha	<i>Håkan Thörngren</i>
GETKA	Get KA from file	File Name in ALPHA	<i>Håkan Thörngren</i>
GETST	Get Status Registers from File	Status File Name in Alpha	<i>Ángel Martin</i>
MKGKA	Merge Assignments	File Name in ALPHA	<i>Håkan Thörngren</i>
RENMF	Rename X-Mem File	"Old, New" Names in ALPHA	<i>Ángel Martin</i>
RETPF	Retype X-Mem File	New type in X, File Name in Alpha	<i>Ángel Martin</i>
RSTCHK	Reset Checksum	Program File Name in Alpha	<i>Håkan Thörngren</i>
SAVEBF	Saves Buffer	Buf id# in X, File Name in Alpha	<i>Håkan Thörngren</i>
SAVEKA	Saves Key Assignments	File Name in ALPHA	<i>Håkan Thörngren</i>
SAVEST	Saves Status Registers	File Name in ALPHA	<i>Ángel Martin</i>
SORTFL	Sort Data File	In ascending order	<i>CCD Module</i>
WORKFL	Gets Work FileName	Appends name to ALPHA	<i>Sebastian Toelg</i>
XQXM	Executes a Program File	Program File Name in ALPHA	<i>Ross Wentworth</i>

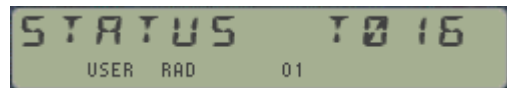
The appendix-2 has a detailed description of the different X-Mem file header structures, which should help to better understand the functionality provided by these functions. The following short descriptions summarize the most important points for each of them:

- **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, RAME** 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"	LIFO
Type	1	2	3	4	5	6	7	8	9	10	15

- **RENMF** is a handy utility that renames an X-Mem file. The syntax is the same used by **RENAME** for the HPIL Disks, that is the string "**OLDNAME, NEWNAME**" must be in alpha. The function will check that the OLDNAME file exists ("**FL NOT FOUND**" condition otherwise), and that there isn't any other file named NEWNAME already ("**DUP FL**" error message).
- **RETPF** 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.
- **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.
- **SORTFL** is a very fast data file sorting function that performs the sorting in ascending order. It is taken from the CCD Module (not easy to extricate its code from the CCD framework). Much faster than equivalent FOCAL routines using GETX/SAVEX or FLRCL/FLSTO.
- **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 CAT4 implementation in the AMC_OS/X module:



A couple of observations are in order:

- The X-Mem file name is expected in ALPHA, thus this imposes a small limitation on things. You can however add a comma to the File Name 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(I-) 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 below.

- Saving and getting KA in/from Extended Memory with **SAVEKA**, **GETKA** and **MRGKA** also expect the File Name in Alpha. **GETKA** will completely replace the existing key assignments with those contained in the file, whilst **MRGKA** will merge them – respecting the unused keys so only the overlapping ones will be replaced. Same error handling is active to avoid file duplication or overwrites. Like their Buffer counterparts they will check for available memory, showing “NO ROOM” when there isn’t enough for the retrieval.
- **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 File Name in Alpha. Error handling includes checking for duplicate buffer (“DUP BF”), buffer existence (“NO BUF”), as well as previous File existence (“DUP FL”).

Sorting Times Comparison.

As an example of utilization, the FOCAL routine below does a bubble-sort using the In-File X-Mem Record functions described later in the manual – same job although in a much slower way than **SORTFL** as seen in the following comparison table:

File Size	FOCAL routine	SORTFL	Speed Factor
10	29.73"	0.43"	69.14x
25	3' 02.19"	1.03"	491.34x
50	12' 03.93"	2.47 "	814.14x
100	50' 45.34"	5.85"	1,446.03x
300	Life is too sort	36.61"	way, way faster
600	Never mind	1' 19.51"	Out of Range ;-)

As you can see including SORTFL in the collection was well worth the effort!

01 LBL "FLSORT"

02 FLSIZE
03 2
04 –
05 E3
06 /
07 STO 00

08 LBL 00

09 RCL 00
10 1.001
11 +
12 STO 01

13 FLRCL IND 00

14 LBL 01

15 FS? 10

16 VIEW 01

17 FLRCL IND 01

18 X>Y?

19 GTO 02

20 FLSTO IND 00

21 X<>Y

22 FLSTO IND 01

23 LBL 02

24 RDN

25 ISG 01

26 GTO 01

27 ISG 00

28 GTO 00

29 CLD

30 END

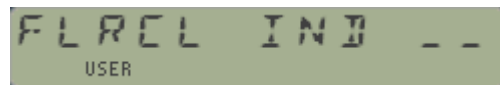
.3.- DATA FILE RECORD FNS

FLRCL __	Recall file record to X	Rec# in prompt / next prgm line	Ángel Martin
FLSTO __	Store X in File Record	Rec# in prompt / next prgm line	Ángel Martin
FLVEW __	View File Record	Rec# in prompt / next prgm line	Ángel Martin
FLX<> __	Swap X and File Record	Rec# in prompt / next prgm line	Ángel Martin
PEEKR	Absolute address RCL	Absolute address in X-reg	Ken Emery
POKER	Absolute address STO	Abs. adr in Y, content in X	Ángel Martin
"DFED	Data File Editor	Data File in ALPHA	Ángel Martin

Starting with the classic ones:

- **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 into 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 whose 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.

The new functions treat the data file records as standard individual registers for store, recall, exchange, and view actions. They are prompting functions: the record number is entered at the function prompt in manual more or taken from the next program line in a running program (non-merged functions again). Obviously, the register number can't be larger than the data file size minus one (zero-base numbering).

For large data files (more than 99 records) you can use the INDirect addressing described below, or alternatively use the Prompt_Lengthener functionality in the AMC_OS/X module to extend the prompt field to three digits:

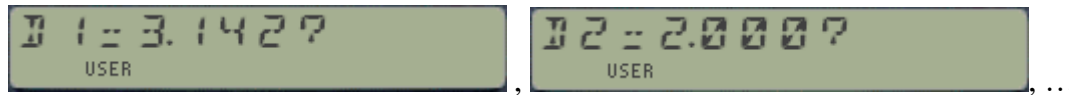



For space reasons the prompting implementation is limited in this module. First, stack addresses and IND stack addresses are not supported. Second, the functions support a hybrid INDirect addressing mode, where the standard data registers are used to hold the file record number – but it's important to realize there's no support for INDirect addressing using the actual file records to hold the pointer to the final data file record.

A complete implementation supporting all types of direct and indirect addressing is available in the XM-TWIN module – refer to its manual for details if you're interested in this functionality.

Data File Editor

The module includes a quick & dirty data file editor – also sorely missing in the standard toolkit. Nothing fancy but a sequential review of the data file records, with the possibility of editing the shown values at each prompt. Press R/S if the shown value doesn't need changing; otherwise input the new value to replace that one shown, and then R/S.



The program listing for **DFED** is shown below. Note that after each prompt the ALPHA register is reset with the file name, thus you can leave the editing at any step – no need to complete the execution till the end of file – very convenient for long files of course.

01 LBL "DFED"	20 FC?C 22 ; entered data?
02 FLSIZE	21 GTO 01 ; no, skip over
03 E	22 X<>Y
04 –	23 RDN
05 E3	24 X<>Y
06 /	25 SEEKPT
07 LBL 00	26 X<>Y
08 SEEKPT	27 SAVEX
09 GETX	28 LBL 01
10 X<>Y	29 X<>Y
11 "D"	30 ISG X(3)
12 ARCL	31 GTO 00
13 >"=	32 "DONE"
14 X<>Y	33 LBL 02
15 ARCL X(3)	34 AVIEW
16 >"?"	35 CLA
17 CF 22	36 WORKFL
18 XEQ 02 ; show & reset	37 END
19 STOP	

Decimal <> Hex Conversions

The module includes two functions for the Dec<->Hex conversions, a classic subject that shouldn't be missing from any utility collection like this one.

D>H and **H>D** perform a quick & dirty conversion between decimal and (unsigned) hex values using the real X-register and ALPHA. Use then when you want to check results independently from the selected base on the emulator.

The maximum number allowed is H: 2540BE3FF or d: 9,999,999,999 in decimal. Both functions are mutually reversed, and **H>D** does stack lift as well.

These functions were written by William Graham and published in PPCJ V12N6 p19, enhancing in turn the initial versions first published by Derek Amos in PPCCJ V12N1 p3.

.4.- LIFO X-FUNCTIONS

Original LIFO functions were written by Doug Wilder and originally published in the BUILD Module.

LIFOINI	Initializes the LIFO Stack	
^A	ALPHA Registers	POPA
^F__	System Flags	POPF
^RTN	RTN Stack Registers	POPRTN
^ST	XYZT Stack	POPST
^X	X-Register	POPX
^Z	X,Y Registers	POPZ

The LIFO (Last In First Out) functions require extended functions memory to operate. The LIFO is located only in the first file in extended memory and must have a minimum size of one register and a maximum size of 120 registers. This structure allows maximum transfer speed, even faster than main memory, and does not require register numbers.

LIFO initialization: Create a first file in extended memory (recommended size is 16 to 32 registers) or if the first file currently in extended memory is of a suitable size, it may be used for the LIFO. Use a sequence similar to: "BUFFER" 28 CRFLD (the name is arbitrary). The function **LIFOINI** converts the first file in extended memory to the LIFO file type, any data in the file is unrecoverable.

The LIFO file type is 15. If you're using the AMC_OS/X Module (always highly recommended), this is shown in a CAT#4 listing with an "L" character in the file type, i.e.:



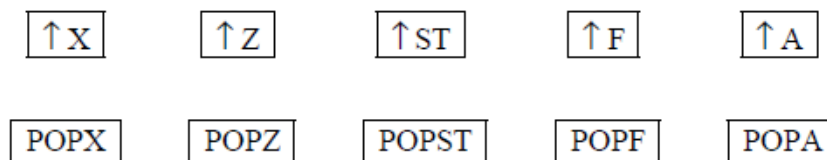
LIFOINI:

Converts the first file in extended memory to LIFO structure and initialize pointers.

After **LIFOINI** has been successfully executed without error, the stack is ready for use. LIFOINI may be executed again to reset the pointers. Ideally, **LIFOINI** would be only executed from the keyboard, however it may also be used in a main program, the uppermost or top driver program.

LIFO functions:

Z: is X and Y (complex data) , T: is Stack (XYZT), F: is Flags, A: is ALPHA, and R: is the RTN stack



If the stack lift is disabled, **POPX** and **POPZ** do not cause a lift, eg, CLX, **POPZ** does not modify the Z and T registers. For multi-register push and pop functions, a "LIFO LIMIT" error leaves the stack in an unknown state and the LIFO pointer is left in an unknown state.

For **POPA** or **POPF**, if a "DATA ERROR" occurs the Alpha/Flag register has not been modified yet the LIFO pointer is left in an unknown state.

Alpha data and Flag data are typed data, that is: one cannot pop numeric or Flag data into Alpha. Stack data is not typed: any type of data may be popped into the XYZT stack.

With a LIFO it is possible to write user code subroutines which simulate monadic functions, for example; do a push stack at entry, put the result in LASTX, then **POPST** and X<>L RTN.

It is also possible to write interrupting alarms which actually do something, they can push the stack/LASTX/Alpha/Flags at entry and recover them at exit. Thank's to HP for the forthought to not interrupt a running program when the stack lift is disabled.

POP of data into the stack is very fast unless a printer is attached, in which case the POP can be greatly slowed due to printer interface. For example a POPST in trace mode will do a full stack printout which can consume up to two seconds. In a running program, clearing F55 will greatly speed things up although trace capability will be lost.

These functions will report "NO XFM LIFO" if a lifo file does not exist. In that case you'll need to create it first and then try again.

Finally, one must remember the basic rule for LIFO stack usage: whatever gets pushed MUST be popped and in reverse order! Otherwise we get what is known as a "memory leak" and eventual LIFO LIMIT error.

Launcher implementation

Besides the individual functions from the summary table, this functionality is also implemented in two LIFO launchers, **ΣPOP** and **ΣPUSH**. Each action is invoking the corresponding POP or PUSH function.

You can use the [SHIFT] key to toggle between them in run mode.



Function	I	A	F	X	Z	T	R
PUSH*	LIFOINI	PUSHA	PUSHF	PUSHX	PUSHZ	PUSHST	PUSHRTN
POP*	LIFOINI	POPA	POPF	POPF	POPZ	POPST	POPRTN

Notice that the option "R" stands for the **POPRTN** and **^RTN** functions, an extension to the original implementation to include the RTN Stack to this scheme.

You can also use the launchers in a program, although the individual functions are a more effective way to operate. In a program each of the options in the launcher prompt needs to be manually added as a second program line, following the launcher function program steps. For instance, the code snippet below saves the contents of the ALPHA register using **PUSHA**:

```
nn      ΣPUSH
nn+1    2
```

.5.- HP-IL RELATED FUNCTIONS

If you haven't noticed, HP-IL related functions are far and in-between but here's a few ones that are frequently needed and sorely missing in the standard ROMS. For example, 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	<i>R. del Tondo</i>
READF	Read Data File	IL FName, XM FName	<i>R.del Tondo</i>
WRTDF	Write Data File	XM FName, IL FName	<i>R. del Tondo</i>
READPG	Reads page from HP-IL	Page# and FileName	<i>Skwid</i>
WRTPG	Writes page to HP-IL	Page# and FileName	<i>Skwid</i>
READXM	Overwrites all XM from IL File	FileName in ALPHA	<i>Skwid</i>
WRTXM	Writes all XM to IL File	FileName in ALPHA	<i>Skwid</i>

- **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 **FLSIZE** 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 FLSIZE
03 ASWAP	10 ASWAP
04 CRFLD	11 CREATE
05 ASWAP	12 ASWAP
06 READF	13 WRTDF
07 RTN	14 END

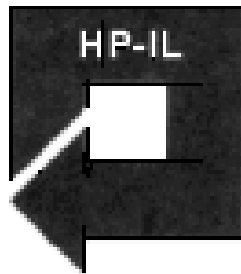
(*) The function **ASWAP** is available in the AMC_OS/X Module, The ALPHA_ROM, and the PowerCL Module amongst other sources.

- **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.

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

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

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



Appendix 1.- 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 2.- 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 linking register of PREVIOUS module (or zero if first block)

NXT: address of top register of NEXT module (or zero if last block)

TOP: address of top register within this module

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

@ 0x301:

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

@ 0x201:

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

@ 0x040:

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

Note: Some of the boundary values appear to be hard-coded in the file management routines, like EMDIR, EMROOM, and file search utilities. This makes it impossible to add more blocks above - even if the memory is available (like is the case for the 41CL machine) - as shown below. it's unfortunately also 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 0.- HP-41 Byte Table

0000 0001 0010 0011 0100 0101 0110 0111								1000 1001 1010 1011 1100 1101 1110 1111								
0 1 2 3 4 5 6 7								8 9 A B C D E F								
0	NULL 00 0	LBL 00 01 1	LBL 01 02 2	LBL 02 03 3	LBL 03 04 4	LBL 04 05 5	LBL 05 06 6	0	LBL 07 08 8	LBL 08 09 9	LBL 09 10 10	LBL 10 11 11	LBL 11 12 12	LBL 12 13 13	LBL 13 14 14	LBL 14 15 15
1	0 16 16	1 17 17	2 18 18	3 19 19	4 20 20	5 21 21	6 22 22	1	8 24 24	9 25 25	10 26 26	11 27 27	12 28 28	13 29 29	14 30 30	15 31 31
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	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 00 48 48	STO 01 49 49	STO 02 50 50	STO 03 51 51	STO 04 52 52	STO 05 53 53	STO 06 54 54	3	STO 08 56 56	STO 09 57 57	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	+ 64 64	- 65 65	* 66 66	/ 67 67	X<Y? 68 68	X>Y? 69 69	X≤Y? 70 70	4	Σ- 72 72	HMS+ 73 73	HMS- 74 74	MOD 75 75	% 76 76	%CH 77 77	P>R 78 78	R>P 79 79
5	LN 80 80	X ² 81 81	SQRT 82 82	Y ^Y 83 83	CHS 84 84	E [↑] X 85 85	LOG 86 86	5	E [↑] X-1 88 88	SIN 89 89	COS 90 90	TAN 91 91	ASIN 92 92	ACOS 93 93	ATAN 94 94	->DEC 95 95
6	1/X 96 96	ABS 97 97	FACT 98 98	X≠0? 99 99	X>0? 100 100	LN1+X 101 101	X<0? 102 102	6	INT 104 104	FRC 105 105	D->R 106 106	R->D 107 107	->HMS 108 108	->HR 109 109	RND 110 110	->OCT 111 111
7	CLΣ 112 112	X<Y 113 113	PI 114 114	CLST 115 115	R [↑] 116 116	RDN 117 117	LASTX 118 118	7	X=Y? 120 120	X≠Y? 121 121	SIGN 122 122	X≤0? 123 123	MEAN 124 124	SEDV 125 125	AVIEW 126 126	CLD 127 127
0 1 2 3 4 5 6 7								8 9 A B C D E F								
8	DEG IND 00 128 128	RAD IND 01 129 129	GRAD IND 02 130 130	ENTER [↑] IND 03 131 131	STOP IND 04 132 132	RTN IND 05 133 133	BEEP IND 06 134 134	8	ASHF IND 08 136 136	PSE IND 09 137 137	CLRG IND 10 138 138	AOFF IND 11 139 139	AON IND 12 140 140	OFF IND 13 141 141	PROMPT IND 14 142 142	ADV IND 15 143 143
9	RCL IND 16 144 144	STO IND 17 145 145	ST+ IND 18 146 146	ST- IND 19 147 147	ST* IND 20 148 148	ST/ IND 21 149 149	ISG IND 22 150 150	9	VIEW IND 24 152 152	Σ REG IND 25 153 153	ASTO IND 26 154 154	ARCL IND 27 155 155	FIX IND 28 156 156	SCI IND 29 157 157	ENG IND 30 158 158	TONE IND 31 159 159
A	XR 0-3 IND 32 160 160	XR 4-7 IND 33 161 161	XR 8-11 IND 34 162 162	XR 12-15 IND 35 163 163	XR 16-19 IND 36 164 164	XR 20-23 IND 37 165 165	XR 24-27 IND 38 166 166	A	SF IND 40 168 168	CF IND 41 169 169	FS?C IND 42 170 170	FC?C IND 43 171 171	FS? IND 44 172 172	FC? IND 45 173 173	STO 46 IND 46 174 174	SPARE IND 47 175 175
B	SPARE IND 48 176 176	GTO 00 IND 49 177 177	GTO 01 IND 50 178 178	GTO 02 IND 51 179 179	GTO 03 IND 52 180 180	GTO 04 IND 53 181 181	GTO 05 IND 54 182 182	B	GTO 07 IND 56 184 184	GTO 08 IND 57 185 185	GTO 09 IND 58 186 186	GTO 10 IND 59 187 187	GTO 11 IND 60 188 188	GTO 12 IND 61 189 189	GTO 13 IND 62 190 190	GTO 14 IND 63 191 191
C	GLOBAL IND 64 192 192	GLOBAL IND 65 193 193	GLOBAL IND 66 194 194	GLOBAL IND 67 195 195	GLOBAL IND 68 196 196	GLOBAL IND 69 197 197	GLOBAL IND 70 198 198	C	GLOBAL IND 72 200 200	GLOBAL IND 73 201 201	GLOBAL IND 74 202 202	GLOBAL IND 75 203 203	GLOBAL IND 76 204 204	GLOBAL IND 77 205 205	X<> IND 78 206 206	LBL IND 79 207 207
D	GTO IND 80 208 208	GTO IND 81 209 209	GTO IND 82 210 210	GTO IND 83 211 211	GTO IND 84 212 212	GTO IND 85 213 213	GTO IND 86 214 214	D	GTO IND 88 216 216	GTO IND 89 217 217	GTO IND 90 218 218	GTO IND 91 219 219	GTO IND 92 220 220	GTO IND 93 221 221	GTO IND 94 222 222	GTO IND 95 223 223
E	XEQ IND 96 224 224	XEQ IND 97 225 225	XEQ IND 98 226 226	XEQ IND 99 227 227	XEQ IND 100 228 228	XEQ IND 101 229 229	XEQ IND 102 230 230	E	XEQ IND 104 232 232	XEQ IND 105 233 233	XEQ IND 106 234 234	XEQ IND 107 235 235	XEQ IND 108 236 236	XEQ IND 109 237 237	XEQ IND 110 238 238	XEQ IND 111 239 239
F	TEXT 0 IND T 240 240	TEXT 1 IND Z 241 241	TEXT 2 IND Y 242 242	TEXT 3 IND X 243 243	TEXT 4 IND L 244 244	TEXT 5 IND M 245 245	TEXT 6 IND N 246 246	F	TEXT 8 IND P 248 248	TEXT 9 IND Q 249 249	TEXT 10 IND R 250 250	TEXT 11 IND a 251 251	TEXT 12 IND b 252 252	TEXT 13 IND c 253 253	TEXT 14 IND d 254 254	TEXT 15 IND e 255 255
0 1 2 3 4 5 6 7								8 9 A B C D E F								
0000 0001 0010 0011 0100 0101 0110 0111								1000 1001 1010 1011 1100 1101 1110 1111								

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.

Copyright 1997, The Museum of HP Calculators