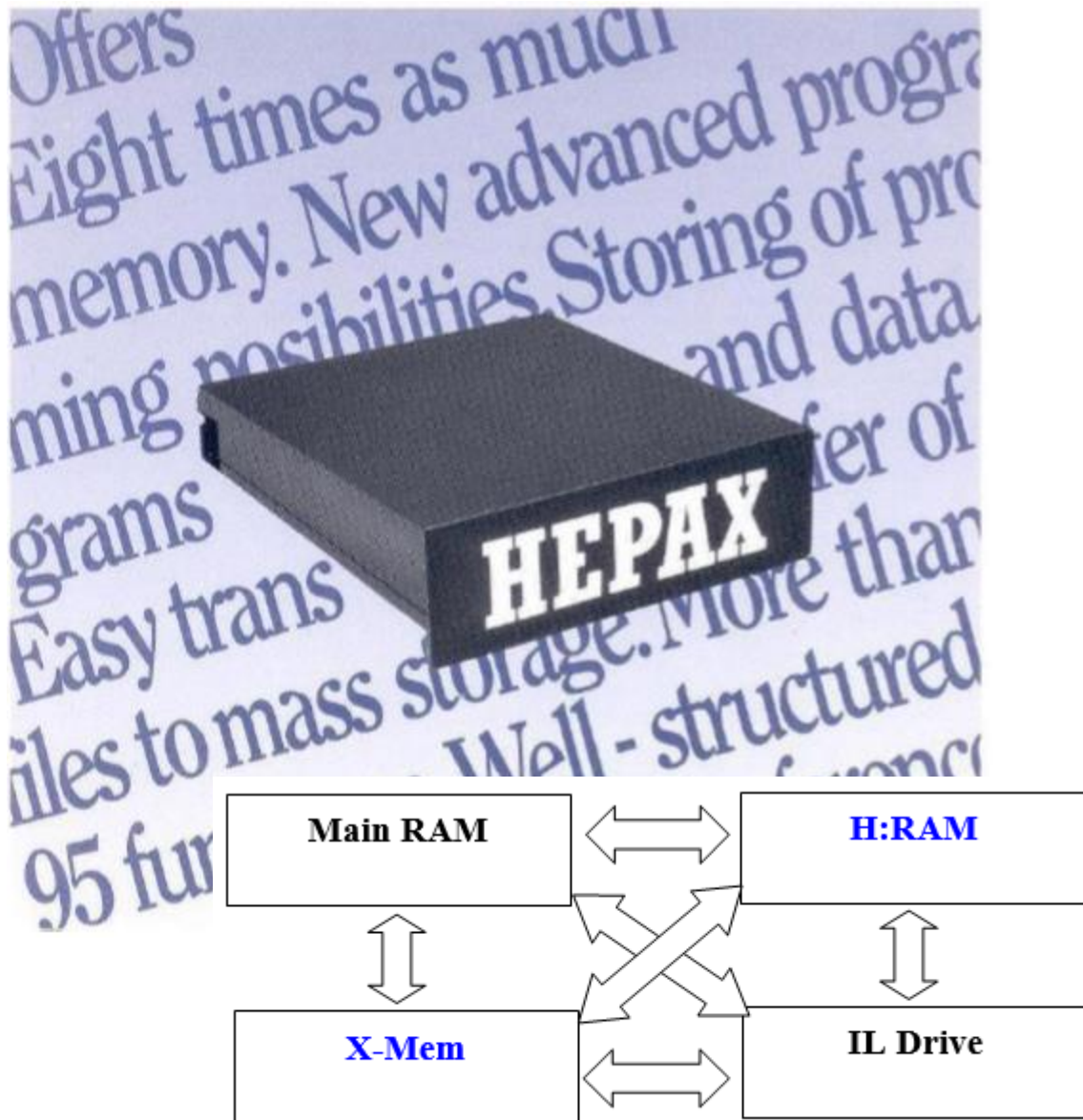


# HEPAX\_4H / HEPAX++ Modules

## Extensions to the HEPAX Platform User's Manual and QRG.

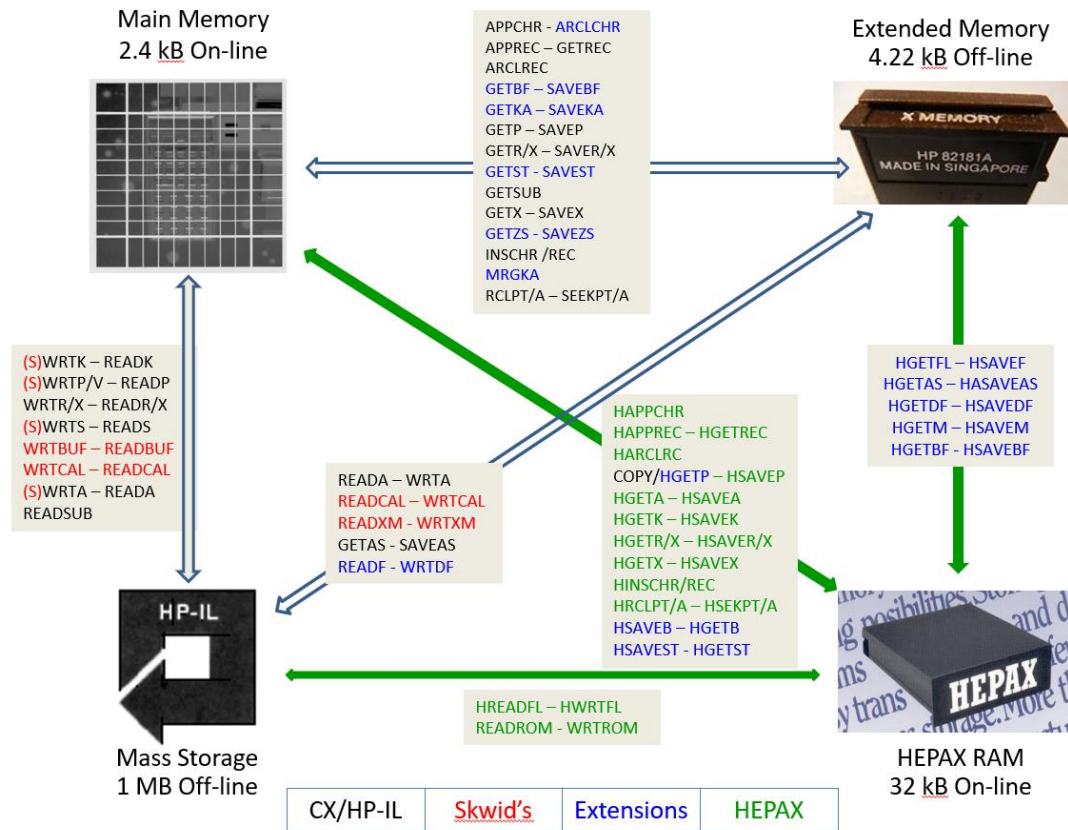


Written and Compiled by Ángel M. Martín

**April 2021**

This compilation revision 1.2.2

Copyright © 2019 -2021 Ángel Martin



Published under the GNU software license 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](http://www.hp41.org)

### Acknowledgments.-

Everlasting thanks to Steen Petersen, original developer of the HEPAX Module – real landmark and seminal reference for the serious MCODER and the 41 system overall. With his product he pushed the design limits beyond the conventionally accepted, making many other contributions pale by comparison.

## HEPAX-4H+ and HEPAX-APPS Modules ORG

### Can a masterpiece be improved?

---

Surely this is a matter of opinion, but most people would respond that there's no need to attempt any improvements on a masterpiece, so why bother then? Perhaps a justification to explain this mini project is that contrary to the art world, engineering lives on constant refinements on existing products. So, it's not so much about improving a certain product but about extending its range of applicability, usefulness or even ease of use.

Whatever the motivation, here's a slightly modified version of the HEPAX Module, put together with the following main considerations in mind:

- Ensure 100% compatibility with the original product. This meant no alterations whatsoever to the FAT, and for sure maintaining all aspects of its functions and functionality.
- Only uses the ROM space left unused in the original product. This imposed the biggest limitation on what could be included and discarded the addition of sub-function catalogs due to their extensive space requirements.
- Just a few additional entries were available in the main FAT and in the sub-function FATs so they have been enlarged within the limits imposed by that restriction.
- Finally, full support of the Library#4 has added the functionality consistent with other modules to include two sub-function catalogs and several other improvements. This has also made more space available, put to good use by the new functions added to the module.

### What's the fuss about?

---

With these criteria in mind, the following additions have been implemented on the Modified HEPAX\_4H:

1. Patched keyboard scanning method for compatibility with the 41CL in function DISASM
2. Removal of the restrictions on the Disassembler and HEX Editor to be used on the HEPAX ROM itself. Although with obvious limitations, such as restricted to the actual bank within the module where the executing code resides.
3. Addition of eight new functions in the main FAT, as follows:

a.	<b>-HEXTRA</b>	-	<b>New Section Header</b>
b.	<b>HFLADR</b>	-	Hepax File Address. File Name in ALPHA
c.	<b>HFLTYP</b>	-	Hepax File Type. File Name in ALPHA
d.	<b>HRTPFL</b>	-	Hepax File Type Change. New type in X.
e.	<b>HWRKFL</b>	-	Gets Hepax Work File Name to Alpha
f.	<b>RLSRAM</b>	-	Releases RAM page from Hepax chain. Page# in X
g.	<b>LSTF</b>	-	Last (sub)function called
h.	<b>CLHM</b>	-	Clears ALL Hepax FileSys

4. Addition of ten new sub-functions to the Auxiliary FATs, as follows:

Accessed via HEPAX# / HEPAX\$:

a.	<b>CAT+</b>	- <b>Sub-function Catalog</b>	HEPAX/A group
b.	<b>CHKSYS</b>	- Checks ROM Configuration	HEPAX/A group
c.	<b>PGROOM</b>	- Calculates available room within page	HEPAX/A group
d.	<b>PGSUM</b>	- Calculates and writes page checksum	HEPAX/A group
e.	<b>ROMLST</b>	- Shows plugged ROMs id's	

Accessed via XF# and XF\$:

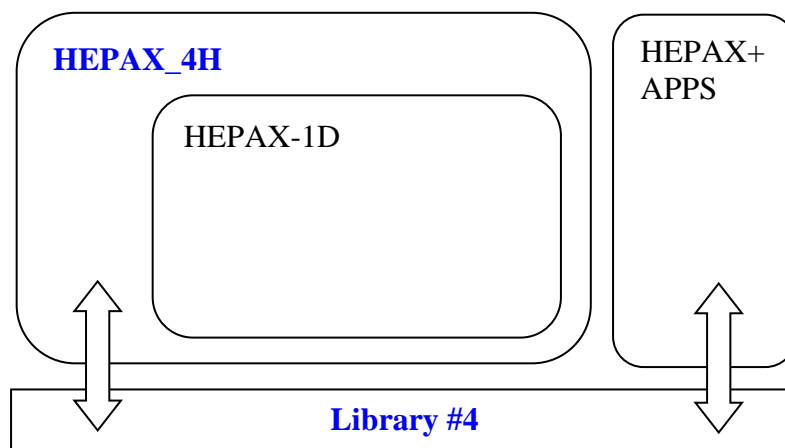
f.	<b>XFCAT</b>	- <b>Sub-function Catalog</b>	XF/A group
g.	<b>CLXM</b>	- Clear Extended Memory	XF/A group
h.	<b>RTN?</b>	- Tests for pending subroutine levels	XF/A group
i.	<b>XQ&gt;GO</b>	- Deletes one pending subroutine return	XF/A group
j.	<b>XQXM</b>	- Executes a program file in X-Memory	XF/A group

The sub-functions catalogs **CAT+** and **XFCAT** feature the same functionality found in other advanced modules. The sub-function enumeration can be stopped, single-stepped, resumed, and access to the displayed sub-function is only a XEQ; away – either in manual or program modes.

Other subtle enhancements included are as follows:

- briefly displaying the sub-function names when using the numeric launchers **HEPAX** and **XF** – this provides useful feedback to the user on which function is called. Note that the name of the alphabetical launchers has been changed to **HEPAX\$** and **XF\$** (i.e. replacing the final "A" in the original with the dollar sign "\$").
- Note as well that the alphabetical launchers **XF\$** and **HEPAX\$** will take the string in Alpha as input *if the user does not enter any alpha chars at the prompt*, terminating the sequence with another ALPHA keypress.

Unfortunately, there was no more room to include a few other functions also related to the HEPAX file system, written by Sebastian Toelg and included in the NEXT ROM. Those are available in the PowerCL, and also included in a derivative project described later on.



## HEPAX Applications ROM

---

An extension of this project is presented below, in the form of a companion ROM that builds on the HEPAX functions. This extension module includes:

- A few left-over functions written by Steen Petersen, taken from several HEPAX pre-production prototypes (see MoHP Forum thread). These also include MCODE print and scan routines, published in the French chapter of PPC.
- Hepax Chain functions from the NEXT ROM (written by Sebastian Toelg) and the PowerCL. They allow selective configuration of the HEPRAM Chain, release of individual blocks, and manual initialization of FileSys pointers.
- Utility functions needed by the User Code routines – to support buffer files and to complement the existing standard set. Notable addition, the function **PCOPY** allows copying of programs in ROM to the main RAM from a running program.
- A comprehensive set of FOCAL routines to manage H:RAM Files, including poor-man Data and ASCII file editors; support for Status, Buffer and Matrix new file types; and exchanges between H:RAM and X-Memory like-to-like files. This group also includes the classic HRESZFL, DISSST and ?JUMP routines from the HEPAX Manuals.

### *HEPAX File Types vs. Other Systems*

---

The table below shows the different file types available in H:RAM (including those created by the Extended-IL ROM) and X-Mem (including those created by advanced modules). Note that they're not always consistent, bear that in mind when using the exchange routines. Note as well that there are no functions available for file exchanges between X-Mem and H:RAM – which is the main theme of this module.

File Type	RAM <-> HEPAX	H:Type	RAM <-> X-Mem	X:Type	RAM <-> HP-IL
Program	HSAVEP - PCOPY	<b>1</b>	SAVEP - GETP	<b>1</b>	WRTP/V – READP
Data	HSAVER/X - HGETR/X	<b>2</b>	SAVEREG/X - GETRG/X	<b>2</b>	WRTR/X – READR/X
ASCII	HAPPREC - HGETREC	<b>3</b>	APPREC - GETREC	<b>3</b>	SAVEAS - GETAS
Matrix	"HSAVEM" - "HGETM"	<b>10</b>	MATDIM, et al.	<b>4</b>	n/a
Key Assignments	HSAVEK - HGETK	<b>5</b>	SAVEKA - GETKA	<b>5</b>	WRTK - READK
Buffer	"HSAVEB" - "HGETB"	<b>7</b>	SAVEBF - GETBF	<b>6</b>	WRTBUB - READBUF
Status Regs	"HSAVEST" - "HGETST"	<b>6</b>	SAVEST - GETST	<b>7</b>	WRTS – READS
Complex Stack	8, "HSAVEB" - "HGETB"	<b>7</b>	SAVEZS - GETZS	<b>8</b>	n/a
LIFO	n/a	-	POP - PUSH	<b>9</b>	n/a
16C Buffer	11, "HSAVEB" - "HGETB"	<b>7</b>	SAVE16 - GET16	<b>11</b>	n/a
ALL	HSAVEA - HGETA	<b>4</b>	n/a	-	WRTA - READA
X-Mem	n/a	<b>9</b>	n/a	-	WRTXM - READXM
CALC(ALL+XM)	n/a	<b>8</b>	n/a	-	WRTCAL - READCAL
ROM	COPYROM	<b>2</b>	n/a	-	WRTROM / READROM

See below the function table, structured in two main sections

-XROM---	ADDR-	FUNCTION ----	Description-----	Type -----	Author -----r
006.00	AFF3	<b>-HEPX_PLUS</b>	Section Header	MCode	n/a
006.01	A1BE	<b>HEPCHN</b>	Make HPX Chain	MCode	Sebastian Toelg
006.02	A137	<b>HEPCHN?</b>	Show HPX Chain	MCode	Sebastian Toelg
006.03	A463	<b>HEPINI</b>	Initialize FileSys	MCode	H. Owen / Á. Martin
006.04	A656	<b>MCPR</b>	Mcode Print	MCode	Steen Petersen
006.05	A7C8	<b>MCSCAN</b>	Mcode Scan	MCode	Steen Petersen
006.06	A546	<b>PCOPY</b>	Programmable COPY	MCode	HP / Á Martin
006.07	A363	<b>RAMTEST</b>	Tests HEPRAM	MCode	Steen Petersen
006.08	A2DC	<b>ROMCHKX</b>	Checks ROM#	MCode	HP Co.
006.09	AB8D	<b>ROMTEST</b>	Tests ROM	MCode	Steen Petersen
006.10	ACF3	<b>READF</b>	Reads Data File to Drive	MCode	R. del Tondo
006.11	A966	<b>WRTDF</b>	Writes Data File from Dr.	MCode	R. del Tondo
006.12	AB7E	<b>FSIZE</b>	IL Drive File Size	MCode	Unlknown
006.13	AFE8	<b>-HEPX_APPS</b>	Section Header	MCode	n/a
006.14	AB82	<b>BFLNG</b>	Buffer Length	MCode	Ángel Martin
006.15	AB8E	<b>BUFHD</b>	Buffer Header	MCode	Ángel Martin
006.16	AB33	<b>CRBUF</b>	Creates Buffer	MCode	Ángel Martin
006.17	ABA9	<b>FLTYPE</b>	File Type	MCode	Ángel Martin
006.18	AB9B	<b>"DISST</b>	SST Disassembly	UCode	Steen Petersen
006.19	AE07	<b>"HASED</b>	H: Text File Editor	UCode	Ángel Martin
006.20	A9BD	<b>"HASVFL</b>	View H:ASCII File	UCode	Ángel Martin
006.21	AFC4	<b>"HCPYAS</b>	Copy H:ASCII File	UCode	Ángel Martin
006.22	AAE1	<b>"HDFED</b>	H: Data File Editor	UCode	Ángel Martin
006.23	A29E	<b>"HGETAS</b>	Gests H:Text to X-Mem	UCode	Ángel Martin
006.24	A609	<b>"HGETB</b>	Gets H:Buffer to X-Mem	UCode	Ángel Martin
006.25	<u>A270</u>	<b>"HGETDF</b>	Gets H:Data to X-Mem	UCode	Ángel Martin
006.26	A259	<b>"HGETFL</b>	Gets HLFile to X-Mem	UCode	Ángel Martin
006.27	<u>AD62</u>	<b>"HGETM</b>	Gets H:Matrix to X-Mem	UCode	Ángel Martin
006.28	ADC5	<b>"HGETST</b>	Gets H:Status	UCode	Ángel Martin
006.29	AECF	<b>"HRESZFL</b>	Resizes H:Text File	UCode	Steen Petersen
006.30	A103	<b>"HSAVEAS</b>	Saves X-ASCII to H:RAM	UCode	Ángel Martin
006.31	A5CC	<b>"HSAVEB</b>	Saves X-Buffer to H:RAM	UCode	Ángel Martin
006.32	A0D4	<b>"HSAVEDF</b>	Saves X-Data to H:RAM	UCode	Ángel Martin
006.33	A0BC	<b>"HSAVEFL</b>	Saves X-Mem File to H:RAM	UCode	Ángel Martin
006.34	AD3B	<b>"HSAVEM</b>	Saves X-Matrix to H:RAM	UCode	Ángel Martin
006.35	AD8E	<b>"HSAVEST</b>	Saves Status to H:RAM	UCode	Ángel Martin
006.36	A054	<b>"HUPDP</b>	Updates Program File	UCode	Werner Huysegoms.
006.37	A637	<b>"?HFT</b>	Auxiliary check	UCode	Ángel Martin
006.38	AE3E	<b>"?JUMP</b>	Jump Distances Coder	UCode	Steen Petersen

### Dependencies.

It comes without saying that all FOCAL routines (and most of the MCODE functions as well) need the **HEPAX\_4H** module plugged in. Make sure you have it and the Library#4 properly installed in your machine.

Some routines will also make use of functions in the **AMC\_OS/X** Module, it's recommended that you have it always plugged as well. Finally, the Matrix File Exchange functions require the **SandMatrix** Module as well.



## **Preamble – General description of the module functions.**

---

### *Functions from HEPAX Prototypes.*

A few MCODE functions are taken from early HEPAX prototypes that were not included in the final release. Thanks to Steen Petersen for making them available via the MoHP Forum. These are: **ROMTEST**, **RAMTEST**, **MCPR**, and **MCSCAN**.

### *HEPAX-Related Functions from other Modules.*

Even if a few routines are also available in other modules, having them in this compilation is the most convenient way to use them. These include **HEPCHN**, **HEPCHN?** And **RLSRAM** from the NEXT\_ROM, written by Sebastian Toelg; as well as **HEPINI** from the PowerCL, originally written by Howard Owen.

### *Other Auxiliary Functions*

For convenience sake, a handful of MCODE functions have also been included to remove the dependencies on other ROMs (specifically the RAMPAGE Module). This is most evident with the Buffer utilities **BUFHD**, **BFHDR** and **CRBUF** but it's also the case with **FLTYPE** and **FLHDR**. All of them are used in several FOCAL routines for file exchanges and management.

### *Routines from the HEPAX Manuals.*

The module includes three routines from the HEPAX manuals that however weren't included in the HEPAX ROM. These are: **HRESZFL**, **DISSST**, and **?JUMP**. You should refer to the HEPAX manuals for description and utilization instructions.

### *So, what's new then, you may ask?*

At the core of the module sits a set of FOCAL routines designed to improve the original HEPAX FileSystem management, including

- support of new file types (Buffer, Status, Matrix)
- allow exchanges with like-to-like files in X-Memory (Program, Data, ASCII, Keys).
- Other routines allow you to clear the complete H:RAM (**HCLFS**) and updating programs in H:RAM keeping the existing FAT order (**HUPDP**, written by Werner Huysegoms).

All in all, I hope you'd agree the module offers a handsome set of utilities and routines to make your HEPAX experience even more enjoying, so go ahead and take it for a spin at the closest H:RAM near you.

### *HEPAX Chain Alteration.*

---

<b>HEPCHN?</b>	Recalls current CHAIN	Placed in ALPHA and Display	Sebastian Toelg
<b>HEPCHN</b>	Sets CHAIN	String in ALPHA	Sebastian Toelg
<b>RLSRAM</b>	Releases RAM page	Pg# in X (decimal)	Sebastian Toelg

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

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



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

Comments:-

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

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

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



*Re-Initializing the Pages control fields.*

<b>HEPINI</b>	Initializes File System	Prompts for values	Author: Howard Owen
---------------	-------------------------	--------------------	---------------------

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

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



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

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

Address	Byte value	Comment
x000	ROM id# => equal to the page#	Always done
xFE7	Previous HRAM page id# (zero if first)	Always done
xFE8	Next HRAM page id# (zero if last)	Always done
xFE9	Fixed value = 091	Won't be overwritten if not zero
xFED	Fixed value = 090	Done always
xFEf	Fixed value = 091	Won't be overwritten if not zero
xFF1	Fixed value = 0E5	Done always
xFF2	Fixed value = 00F	Done always
xFF3	Fixed value = 200 (or 100)	Done always

Two of the byte values shown in the table above located at addresses 0xpFE9 and 0xpFEF have a different treatment: they will be branded only if their current content is zero. They denote the initial address in the page where the next file or program will be written using **HSAVEP**, **HCRFLAS** and **HCRFLD**. Their values will vary as more programs or content is written to the HRAM page, thus should not be overwritten by **HEPINI** – or else the HEPAX FileSys catalog will become corrupt.

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

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

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



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

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

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

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

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

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

### *New H:RAM File Types*

---

<b>HSAVEB</b>	Save Buffer to H:RAM	Buf id# in X, FileName in ALPHA	FileType 7
<b>HGETB</b>	Get buffer from H:RAM	File Name in ALPHA	
<b>HSVEST</b>	Save Status Regs in H:RAM	File Name in ALPHA	FileType 4
<b>HGETST</b>	Get Status Regs from H:RAM	File Name in ALPHA	

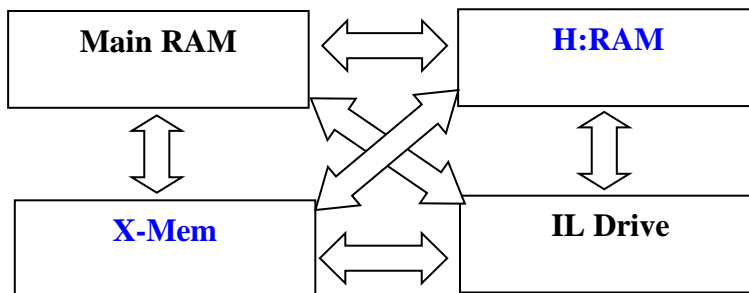
These functions extend the H:RAM support to two new file types, dedicated to storing the contents of I/O Buffers and the Status regs - from T(0) to Q(9) only. Interestingly, the HEPAX\_4H will display these files with their own code (BF and ST), so the HEPDIR enumeration will show a properly labeled code for them. How is that so, you may ask? This is a collateral effect derived from the **Extended-IL ROM**, which is indeed capable of creating these file types on the HP-IL Drive. Add to that the action of the HEPAX functions **WRTFL** and **READFL** and you have a nice way to save those files in the HLRAM FileSystem as well.

Inspired by such an scenario the FOCAL routines in this module can be used to store/recall I/O buffers and sets of Status Registers in H:RAM. Initially the new files have DATA format, so we can use the standard **GETX/SAVEX** commands on them – but thanks to the **HRTPL** function the file type is changed to the special ones at the end and the beginning of the programs, tricking the user into believing they have full support, so to speak.

Notice that if the I/O buffer already exists in RAM, **HGETB** will override its content (and resize it appropriately) so it can be considered a buffer update. Apart from this the remaining routines are self-explanatory as well, just mind the duplicate files and wrong types on input.

Note: this is not to be confused with the buffer and status regs files that can also exist in X-Memory, managed by the OS/X and other advanced modules. So here we're not moving between X-Mem and H:RAM... don't lose your bearings just yet ;-)

The sketch below shows the possible exchanges between the four main data repositories in the HP-41 System. Many functions exist in this and other modules to do the swapping and copying...



And now that we're on this subject, have you ever noticed the mis-naming convention used by HP in the X-Functions module with the **GETAS** and **SAVEAS** functions? Since they move data between X-Mem and the HP-IL Drive, they should have been called **WRTAS** and **READAS** instead. We don't know who we have to thank for that blunder but certainly will remain as a source of confusion forever... names should matter!

Program listing.-

01	LBL "HSAVEB"	bf id# in X	36	LBL "HGETB"	
02	BFLNG		37	7	
03	2		38	XROM "?HFT"	
04	+		39	2	
05	HCRFLD		40	HRTPL	
06	,		41	CLX	
07	HSEKPTA		42	HSEKPTA	
08	R^		43	HGETX	bf id#,SZE to X
09	R^	bf id# in X	44	B?	
10	BFLNG		45	CLB	
11	E3		46	CRBUF	
12	/		47	BUFHD	
13	+	id,00(sz)	48	LBL 01	
14	HSAVEX		49	SF 25	
15	BUFHD	buf address	50	HGETX	
16	PEEKR	buf header	51	POKER	
17	HSAVEX	buf header	52	RDN	
18	RDN	buf address	53	E	
19	E		54	+	
20	-		55	FC? 25	
21	X<>Y	id,00(sz)	56	GTO 01	
22	INT	bf id# in X	57	7	
23	BFLNG		58	HRTPL	
24	RCL Z	length in Y, addr in X	59	RTN	
25	LBL 00		60	LBL "?HFT"	
26	PEEKR		61	HFLTYP	
27	HSAVEX		62	X=Y?	
28	RDN		63	RTN	
29	E		64	CLX	
30	+	next bf reg address	65	E	
31	DSE Y	decrease counter	66	X=Y?	`program file?
32	GTO 00		67	HGETX	yes, force error
33	7		68	ST+ X	
34	HRTPL		69	X=Y?	data file?
35	RTN		70	HGETREC	yes, force error
			71	HGETX	force error
			72	END	

Obvious use of the buffer functions in here, both from the AMC\_OS/X and from this module as well - now you know why they were included in the module. Note also the utilization of functions **PEEKR** and **POKER** from the AMC\_OS/X Module, whose mission is to read/write the buffer registers to/from the X-register

Also, the observant reader would have noticed that the H:RAM buffer file type is **7**. Hold on to this fact for a later discussion on file type numbers across the systems.

Program listing (Con't).-

<b>1</b>	<b>LBL "HSAVEST"</b>		<b>27</b>	<b>LBL "HGETST"</b>	
2	STO Q(9)	<i>destroys Q !</i>	28	4	
3	CLX		29	XROM "?HFT"	
4	E1		30	2	
5	HCRFLD		31	H RTPFL	
6	CLX		32	6	
7	HSEKPTA		33	HSEKPTA	
8	X<> Q(9)	<i>recovers X</i>	34	5,009	
9	R^		35	LBL 01	
10	HSAVEX	<i>saves T</i>	36	HGETX	
11	E^		37	POKER	
12	HSAVEX	<i>saves Z</i>	38	RDN	
13	R^		39	ISG X	
14	HSAVEX	<i>saves Y</i>	40	GTO 01	
15	R^		41	LBL 10	
16	HSAVEX	<i>saves X</i>	42	,	
17	X<> L(4)		43	HSEKPTA	
18	HSAVEX	<i>saves L</i>	44	HGETX	
19	5,009		45	HGETX	
20	LBL 00		46	HGETX	
21	PEEKR		47	HGETX	
22	HSAVEX		48	X<> L(4)	
23	EDN		49	HGETX	
24	ISG X		50	X<> L(4)	
25	GTO 00		51	STO 9(Q)	
26	GTO 10	<i>restores T-L</i>	52	CLX	
			53	4	
			54	H RTPFL	
			55	X<> Q(9)	
			56	END	

Here the major nuisance is to avoid overwriting the original contents of the stack registers due to the actual use of them by the running program – which makes us resort to a couple of tricks, even if the program length suffers a bit. Thankfully we can reuse part of the restore code in the LBL 10 subroutine for both the saving and the restoring cases.

PS. I should point out that the Q register is irredeemably lost, but that's inevitable when using FOCAL code since Q is used as scratch in multiple functions.

Also note that the Status File in H:RAM has type = 4

### *Exchanging X-Mem and H:RAM like-to-like Files*

<b>HSAVEFL</b>	Save X-Mem file to H:RAM	File Name in ALPHA	FileTypes: 1,2,3,4,5,8
<b>HGETFL</b>	Get H:RAM File to X-Mem	File Name in ALPHA	
<b>HSAVEDF</b>	Save X-Mem DataFile in H:RAM	File Name in ALPHA	FileType 2
<b>HGETDF</b>	Get H:RAM DataFile to X-Mem	File Name in ALPHA	
<b>HSAVEAS</b>	Save X-Mem DataFile in H:RAM	File Name in ALPHA	FileType 3
<b>HGETAS</b>	Get H:RAM DataFile to X-Mem	File Name in ALPHA	
<b>HSAVEM</b>	Save Matrix in H:RAM	File Name in ALPHA	FileType 8
<b>HGETM</b>	Get H:RAM Matrix to X-Mem	File Name in ALPHA	

These routines provide a convenient method to exchange files between X-Memory and the H:RAM FileSystem. All of them follow a similar design, whereby the file name is expected in ALPHA and will be the same in both storage areas. The "SAVE" in the name implies moving it to the H:RAM, whereas "GET" means the opposite direction, i.e. moving it back to X-Memory.

Strictly speaking, the Matrix Case will not only support X-Memory Matrix files but also matrices stored in main RAM (and even the CL Y-registers), but that's just an additional functionality controlled by the matrix name as you know ("R" or "Y" followed by a number to denote those cases respectively). An extra bonus for you.

As you can see, we have dedicated global labels for DATA, ASCII and MATRIX types – but not so for Program and Keys files, right? This is done to avoid confusion with already existing functions such as HSAVEP and HSAVEK (and their "GET" counterparts of course). For these cases we have the general-purpose **HGETFL** and **HSAVEFL**, which automatically determine the specific type using the **HFLTYP** function.

As always, make sure there's not already a file with the same name in the destination location or you'll get the "H:DUP FL" error message.

### *Like-to-Like File Exchange Routines.*

The tables below show the routines used to exchange like-to-like files between (a) X-Mem and H:RAM, and (b) X-Mem and HP-IL.

File type	→ HEPAX	→ X-Mem	File type	→ HP-IL	→ X-Mem
Program	"HSAVEFL"	"HGETFL"	Program	<b>WRTFL</b>	<b>READFL</b>
Data	"HSAVEDF"	"HGETDF"	Data	WRTDF	READF
ASCII	"HSAVEAS"	"HGETAS"	ASCII	SAVEAS	GETAS
KEYS	"HSAVEFL"	"HGETFL"	KEYS	<b>WRTFL</b>	<b>READFL</b>
MATRIX(*)	"HSAVEM"	"HGETM"	X-MEM	<b>WRTXM</b>	<b>READXM</b>

(\*) Requires SandMatrix

Note that for Data and ASCII files with the HP-IL as destination the file must already exist in the drive and be of the same size as in the source, so technically only the contents are transferred. The Extended-IL ROM overcame this limitation with the more powerful **WRTFL/READFL** and **WRTXM/READXM** functions.



Program listing/-

01	LBL "HSAVEFL"	<u>X-Mem to H:RAM</u>	01	LBL "HGETFL"	<u>H:RAM -&gt; X-Mem</u>
02	FLTYPE		02	HFLTYP	
03	GTO IND X		03	GTO IND X	
04	LBL 01	<u>PROGRAM</u>	04	LBL 01	<u>PROGRAM</u>
05	GETP	read program	05	PCOPY	Copy Program
06	HSAVEP	save in H:RAM	06	SAVEP	save in X-Mem
07	PCLPS	clear from RAM	07	PCLPS	clear from RAM
08	RTN	done.	08	RTN	done.
09	LBL 02	<u>DATA</u>	09	LBL 02	<u>DATA</u>
10	LBL "HSAVEDF"	<u>X-Mem to H:RAM</u>	10	LBL "HGETDF"	<u>H:RAM -&gt; X-Mem</u>
11	SIZE?	current size	11	SIZE?	current size
12	FLSIZE	file size	12	HFLSIZE	file size
13	X>Y?	larger?	13	X>Y?	larger?
14	PSIZE	yes, resize	14	PSIZE	yes, resize
15	HCRFLD	create counterpart	15	CRFLD	create counterpart
16	,	beginning	16	,	beginning
17	SEEKPTA	put X-pointer	17	HSEKPTA	put H:Pointer
18	HSEKPTA	put H:pointer	18	SEEKPTA	put X-Pointer
19	GETR	read file	19	HGETR	read data
20	X<>Y	size to X	20	X<>Y	size to X
21	E		21	E	
22	-	starts in R00	22	-	starts in R00
23	E3		23	E3	
24	/	R00 - R(N-1)	24	/	R00 - R(N-1)
25	HSAVERX	copy registers	25	SAVERX	copy registers
26	RTN	done.	26	RTN	done.
27	LBL 04	<u>MATRIX</u>	27	LBL 08	<u>MATRIX</u>
28	XROM "HSAVEM"		28	XEQ "HGETM"	
29	RTN		29	RTN	
30	LBL 05	<u>KEYS</u>	30	LBL 05	<u>KEYS</u>
31	GETKA		31	HGETK	
32	HSAVEK		32	SAVEKA	
33	END		33	END	
34	LBL 03	<u>ASCII</u>	34	LBL 03	<u>ASCII</u>
35	LBL "HSAVEAS"	<u>X-Mem to H:RAM</u>	35	LBL "HGETAS"	<u>H:RAM -&gt; X-Mem</u>
36	FLSIZE	file size	36	HFLSIZE	file size
37	HCRFLAS	create counterpart	37	CRFLAS	create counterpart
38	CLX	beginning	38	CLX	beginning
39	SEEKPTA	put X-pointer	39	HSEKPTA	put H:Pointer
40	HSEKPTA	put H:pointer	40	SEEKPTA	put X-Pointer
41	LBL 00		41	LBL 00	
42	SF 25	set error flag	42	SF 25	set error flag
43	GETREC	get record	43	HGETREC	get record
44	FC?C 25	out of bounds?	44	FC?C 25	out of bounds?
45	RTN	yes, all done.	45	RTN	yes, all done
46	HAPPREC	no, append to file	46	APPREC	no, append record
47	LBL 10		47	LBL 10	
48	FC? 17	full record?	48	FC? 17	full record?
49	GTO 00	yes, do next	49	GTO 00	yes, do next
50	GETREC	no, get remainder	50	HGETREC	no, get remainder
51	HINSREC	insert in file	51	INSREC	insert in file
52	GTO 10	do next	52	GTO 10	do next
53	END		53	END	

---

Program remarks.

---

1.- A couple of observations on the code above pertain to the use of the SAVEK/GETK functions in the AMC\_OS/X to manage *Key Assignment files* in X-Memory. With them the Keys file type is a trivial exercise, simply pairing them with the HEPAX-equivalent functions HSAVEK/HGETK. This requires actual re-assignment of the keyboard in the process, be aware of this fact so you won't wonder where the original key assignment went.

2.- The *Data Files* use a similar approach, making the transfer "in masse" with the block functions GETR/SAVER – instead of using a one-by-one register with GETX/SAVEX. It's faster this way but requires that sufficient SIZE be allocated in the calculator, which may not be possible for very large DATA files of course. See below for an alternative routine using the one-record at a time approach.

<b>1</b>	<b>LBL "HGETDF2"</b>		<b>15</b>	<b>LBL "HSAVDF2"</b>	
2	HFLSIZE	H:RAM FileSize	16	FLSIZE	X-Mem size
3	CRFLD	create X-Mem file	17	HCRFLD	create H:RAM file
4	,	start record	18	,	start record
5	HSEKPTA	put H-pointer	19	HSEKPTA	put H-pointer
6	SEEKPTA	put X-pointer	20	SEEKPTA	put X-pointer
7	LBL 00		21	LBL 01	
8	SF 25	set error flag	22	SF 25	set error flag
9	HGETX	get H-value	23	GETX	get X-value
10	FC?C 25	last one?	24	FC?C 25	last one?
11	RTN	yes, done	25	RTN	yes, done
12	SAVEX	write to X-Mem	26	HSAVEX	write to H:RAM
13	GTO 00	no, do next	27	GTO 01	do next
14	RTN		28	END	

3.- *Matrix files* are handled in a separate routine called from the main one. See below the listing. Note the obvious use of the matrix functions from the SandMatrix module:

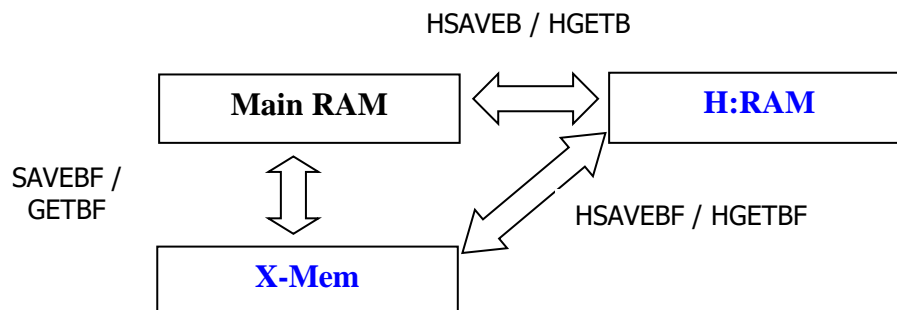
<b>01</b>	<b>LBL "HSAVEM"</b>	Save in H:RAM	<b>20</b>	<b>LBL "HGETM"</b>	Get from H:RAM
02	FLSIZE	get file size	21	8	target type
03	E	bump by one	22	XROM "?HFT"	check for it
04	+	to save dimension	23	2	
05	HCRFLD	create DATA file	24	HRTPL	re-type to DATA
06	,	start of matrix	25	,	start of file
07	MSJIA	put pointer	26	HSEKPTA	put pointer
08	HSEKPTA	put file pointer	27	HGETX	get dimension
09	DIM?	recall dimension	28	MATDIM	dimension the matrix
10	HSAVEX	save in file	29	,	start of matrix
11	LBL 00		30	MSJIA	put pointer
12	MRR+	recall next element	31	LBL 01	
13	HSAVEX	save in file	32	HGETX	get next element
14	RDN		33	MSR+	save in matrix
15	FC? 10	out of bounds?	34	RDN	
16	GTO 00	no, do next	35	FC? 10	out of bounds?
17	8	yes, target type	36	GTO 01	no, do next
18	HRTPL	re-type file	37	8	yes, target type
19	RTN	done	38	HRTPL	re-type to MATRIX
			39	END	done.

4.- Also remark the use of **PCOPY** for the *Program files*, paired with HSAVEP to achieve a clean and efficient program transfer between both media – if it only requires that the program be in main RAM as intermediate step, which may introduce some practical limitations.

5.- Conspicuously absent is the case of *Buffer file transfer*, which indeed poses an interesting challenge for User code. You would think that shouldn't be the case, since the AMC\_OS/X module is equipped with SAVEB/GETB, so together with PEEKR/POKER will certainly fit the need, right? The issue lies in getting the buffer id# from its binary value in the buffer header into a proper BCD value in X-register. This made the program length long enough to exceed the available room in the ROM, thus it wasn't included in the module. The routine below closes the gap, should you be in need for it.

1	LBL "HSAVEBF"	X-Mem Name in ALPHA	17	LBL "HGETBF"	H:RAM FileName in ALPHA
2	GETB	write to I/O	18	XROM "HGETB"	write to I/O RAM
3	BUFHD	get header address	19	2	data type
4	PEEKR	get header register	20	HRTPL	change type
5	DCD	decode it	21	CLX	beginning of file
6	ATOX	A - F = 65 - 70	22	HSEKPT	put pointer
7	64	first ALPHA char	23	HGETX	buf id# in X
8	X<=Y?	is it A - J ?	24	SAVEB	save in X-Mem
9	GTO 00	yes, skip over	25	7	buf type
10	CLX	1 - 9 = 49 - 57	26	HRTPL	retype
11	48	offset mask	27	END	done
12	LBL 00				
13	-	buf id# in X			
14	WORKFL	in DM-41X			
15	XROM "HSAVEB"	save buffer to H:RAM			
16	RTN	done.			

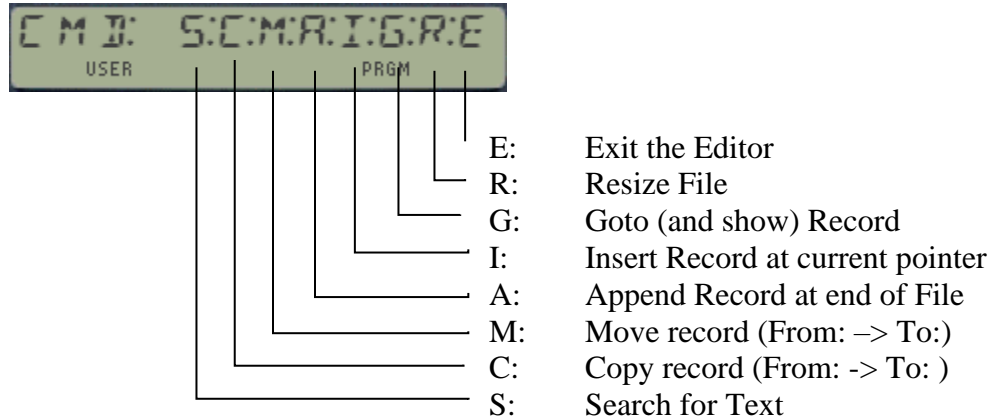
The sketch below shows the six functions and how they apply to the different systems:



**"HASED" - ASCII File Editor**

X: File Size (if new); ALPHA: File Name

**HASED** is a poor-man's ASCII file Editor for the H:RAM File System. Its main interface presents a choice of commands to execute actions on the current record or on the file itself, as follows:



The File may be new or exist previously; the editor will allow for either case but if it's new then the file size needs to be entered X as well as the name in ALPHA.

The Editor checks for ASCII File type if it already exists, showing the customary "H:FL TYPE ERR" error message when that condition is not met.

The *Add Record* command will prompt "TEXT:" for you to enter the text string, max. 24 chars. Then press R/S to input the text and repeat the process until no more text is needed, so R/S is pressed with no text introduced.

The *Resize File* command allows you to change the size of the DATA and Text files. The file can be upsized or downsized, in case more text records need to be added. Two special circumstances are:

- If you input the same file no call to HRESZFL will take place, and
- If you press R/S at the "SIZE=?" prompt the file will be downsized again eliminating the unused empty records that may exist.

The *Move and Copy Record* commands use a "From: - To:" approach to select which record news to be moved or copied ("FROM:") and to where ("TO:"). Both need to be within the file size quite obviously, and the record will be inserted at the specified record number.

Use the *Search* command to position the file pointer at the (first) record containing the string entered at the "TEXT:" prompt. The search will start at the beginning of the file (i.e. record #0). If not found the prompt will appear again, press R/S to return to the command choices.

The other options are self-explanatory in the summary above, and don't require any special considerations.

Note that **HASED** uses functions from the AMC\_OS/X Module (PMTA, PMTK, ARCLI) - therefore ensure that it's also plugged in the calculator.

Program listing.-

1	<b>*LBL "HASED"</b>	<b>; ASCII Files</b>	50	<b>*LBL 01</b>	
2	SF 25	; set error flag	51	CF 23	
3	HFLSIZE	; get file size	52	CLA	
4	FC?C 25	; already exists?	53	PMTA	
5	HCRFLAS	; no, create it	54	FC? 00	
6	3	; must be ASCII	55	HAPPREC	
7	XROM "?HFT"	; check type	56	FS? 00	
8	*LBL 00	; MENU	57	HINSREC	
9	<b>"CMD: SCMAIGRE"</b>		58	FS?C 23	
10	PMTK	; input choice	59	GTO 04	
11	GTO IND X	; divert there	60	GTO 00	
12	<b>*LBL 01</b>	<b>; SEARCH Text</b>	61	<b>*LBL 06</b>	<b>; GOTO Rec</b>
13	CLA	; clear ALPHA	62	"REC#=?"	
14	PMTA	; input text	63	PROMPT	
15	,	; start of file	64	SF 25	
16	HSEKPT	; put pointer	65	HSEKPT	
17	HPOSFL	; search for string	66	FC?C 25	
18	>"?"		67	GTO 06	
19	X<0?	; found?	68	HGETREC	
20	AVIEW	; no, show	69	AVIEW	
21	X<0?		70	HSEKPT	
22	PSE	; and pause	71	GTO 00	; To Menu
23	GTO 00	; go to Menu	72	<b>*LBL 07</b>	<b>; RESIZE File</b>
24	<b>*LBL 02</b>	<b>; COPY Record</b>	73	HFLSIZE	
25	XEQ 10		74	CF 22	
26	X<>Y		75	"SIZE=?""	
27	HSEKPT		76	PROMPT	
28	GTO 00	; Go To Menu	77	FC? 22	
29	<b>*LBL 03</b>	<b>; MOVE Record</b>	78	GTO 07	
30	XEQ 10		79	X#Y?	
31	RCL 00		80	XROM "HRESZFL"	
32	RCL Z		81	GTO 00	; To Menu
33	X<Y?		82	<b>*LBL 07</b>	
34	ISG Y		83	CLA	
35	""		84	HASROOM	
36	X<>Y		85	7	
37	HSEKPT		86	/	
38	HDELREC		87	INT	
39	X<>Y		88	-	
40	X>Y?		89	HWRKFL	
41	DSE X		90	XROM "HRESZFL"	
42	""		91	RTN	
43	HSEKPT		92	<b>*LBL 08</b>	<b>; EXIT Editor</b>
44	GTO 00	; To Menu	93	HWRKFL	
45	<b>*LBL 04</b>	<b>; ADD Rec</b>	94	RTN	
46	CF 00		95	<b>*LBL 10</b>	<b>; From/To</b>
47	GTO 01		96	HRCLPT	
48	<b>*LBL 05</b>	<b>; INSERT Rec</b>	97	"FROM:"	
49	SF 00		98	PROMPT	

99	HSEKPT	123	HSEKPT
100	STO 00	124	X<>Y
101	"TO:"	125	E3
102	PROMPT	126	/
103	E	127	R^
104	*LBL 09	128	+
105	HGETREC	129	SF 17
106	ALENG	130	*LBL 12
107	6 E-5	131	CLA
108	+	132	ARCL IND X
109	E	133	ISG X
110	*LBL 13	134	ARCL IND X
111	ASTO IND Z	135	ISG X
112	ASHF	136	ARCL IND X
113	ST+ Z	137	ISG X
114	DSE Y	138	ARCL IND X
115	GTO 13	139	FS? 17
116	R^	140	HINSREC
117	R^	141	FC?C 17
118	FS? 17	142	HAPPCHR
119	GTO 09	143	ISG X
120	E	144	GTO 12
121	-	145	END
122	X<>Y		
146			



**"HASVFL" – View ASCII File**

ALPHA: File Name

**HASVFL** has been promoted into its own, separate routine (it used to be part of the Editor). This short routine sequentially shows all records starting from the beginning and leaves the file pointer at the end. The contents are preceded by a short indication with the record number, so you know its exact whereabouts within the file.



## Program Listing.-

<pre> 01 *LBL "HASVFL" 02 , 03 HSEKPTA 04 *LBL 11 05 "#" 06 E 07 HRCLPT 08 X#0? 09 + 10 ARCLI 11 &gt;"," 12 AVIEW 13 PSE </pre>	<pre> 14 SF 25 15 HGETREC 16 FC?C 25 17 GTO 00 18 AVIEW 19 12 20 ALENG 21 X&lt;=Y? 22 PSE 23 GTO 11 24 *LBL 00 25 HWRKFL 26 END </pre>
---	--

**"HCLFS" – Clear H:RAM FileSystem**

**HCLFS** is a minimalistic routine that deletes all files from the H:RAM FileSystem, one by one in a sequential fashion. Use it to reset your RAM pages without changing the CHAIN pointers (as would occur if you used CLRAM, for instance).

## Routine listing:

```

01 *LBL "HCLFS"
02 *LBL 00
03 E
04 HEPDIRX
05 SF 25
06 HPURFL
07 FS?C 25
08 GTO 00
09 END

```

**"HCPYAS" – Copy ASCII File**

ALPHA: "FROM,TO" File Names

**HCPYAS** is a new addition to the module. As its name implies, you can use it to make an exact copy of an ASCII file into another, just write the "FROM,TO" file names in ALPHA and execute the routine.

If the destination file already exists, it'll be purged and recreated to make sure it has the same size and no contents when the routine begins.

The routine does not use any data registers, only the stack. Note how the files are selected using the HUNSEC function, chosen because it does not return any value to the stack (which is already holding the record contents or the "FROM,TO" string.

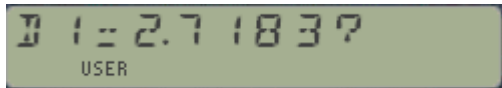
Program listing.-

1	<b>*LBL "HCPYAS"</b>	21	RTN
2	HFLSIZE	22	XEQ 01
3	ASWAP	23	ASWAP
4	SF 25	24	HUNSEC
5	HPURFL	25	XEQ 01
6	CF 25	26	HAPPREC
7	HCRFLAS	27	XEQ 01
8	.	28	ASWAP
9	HSEKPTA	29	GTO 02
10	ASWAP	30	<b>*LBL 01</b>
11	HSEKPTA	31	X<> P
12	<b>*LBL 02</b>	32	RDN
13	RCL M	33	X<> O
14	RCL N	34	RDN
15	RCL O	35	X<> N
16	RCL P	36	RDN
17	HUNSEC	37	X<> M
18	SF 25	38	RDN
19	HGETREC	39	END
20	FC?C 25		

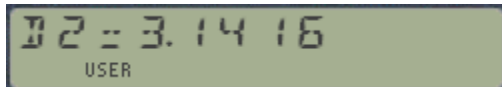
**"HDFED" - ASCII File Editor**

ALPHA: File Name

**HDFED** is the counterpart editor routine for DATA files in the H:RAM File System. It's however less feature-rich and thus much simpler to use, offering just a sequence of prompts showing the current values starting from the beginning of the file. At each prompt you can enter a new value to update the file record if user flag F8 is clear, or press R/S to keep the current values as shown in the prompt. Setting F8 will only do a review of the values without editing capability



Note the "?" with F8 clear



with F8 set (no editing)

Note that contrary to **HASED**, the data file must exist already when you call this poor-man's editor routine – or else the "**H:FL NOT FOUND**" error will be shown. You can always use the **HRESZFL** routine manually to adjust the file size as appropriate.

You can leave the editor at any time but note that the file name won't be in ALPHA until you review all file records. If you need it back you can use function **HWRKFL** in the HEPAX\_4H module.

## Program listing.-

1	<b>*LBL "HDFED" ; DATA Files</b>	21	GTO 02
2	HFLSIZE	22	FS? 08
3	E	23	GTO 01
4	-	24	X<>Y
5	E3	25	RDN
6	/	26	X<>Y
7	*LBL 00	27	HSEKPT
8	HSEKPT	28	X<>Y
9	HGETX	29	HSVEX
10	X<>Y	30	*LBL 02
11	"D"	31	X<>Y
12	ARCLI	32	*LBL 01
13	>"="	33	ISG X
14	X<>Y	34	GTO 00
15	ARCL X	35	"DONE"
16	CF 22	36	AVIEW
17	FC? 08	37	CLA
18	>"?"	38	HWRKFL
19	PROMPT	39	END
20	FC?C 22		

**"HUPDP" – Update Program**

ALPHA: Program Name

Editing a program saved in HLRAM requires copying it to RAM first, and then saving it back to H:RAM. Unfortunately, the standard behavior during HSAVEP \*moves\* the program file from its initial location within the chain (and the FAT) to the end of the list. This is a problem if the routine was called from other user programs, because the XROM id# gets changed.

Written by Werner Huysegom, **HUPDP** allows you to update a program in the H:RAM FileSystem without the undesired effect of changing its FAT location in the process, and therefore maintaining compatibility with any potential user code (both in RAM and in H:RAM). This routine takes advantage of the programmable Copy function PCOPY, coupled with HSAVEP to move those program files in H:RAM

Limitations: Note that PCLPS will purge all programs located after the one being updated, should that be the case. Also, because the program name it's saved using ASTO, it can't be seven characters long. This restriction could be easily removed using RCL M/N instead of ASTO X/Y...

Program listing.-

1	<b>*LBL "HUPDP"</b> ; Update PRGM	31	X#Y?
2	ASTO X	32	GTO 00
3	>" " ; add 5 blanks	33	ASTO X
4	ASTO Y	34	R^
5	E	35	X=Y?
6	*LBL 02	36	RTN
7	HEPDIRX	37	RCL Z
8	0	38	32
9	X=Y?	39	ENTER^
10	GTO 00	40	*LBL 04
11	ASTO X	41	CLX
12	R^	42	-1
13	X#Y?	43	AROT
14	ISG L	44	CLX
15	GTO 01	45	ATOX
16	R^	46	X=Y?
17	LASTX	47	GTO 04
18	GTO 02	48	XTOA
19	*LBL 01	49	<b>PCOPY</b>
20	RDN	50	HSAVEP
21	CLX	51	PCLPS
22	LASTX	52	RDN
23	*LBL 00	53	GTO 01
24	CLA	54	*LBL 00
25	ARCL Z	55	ISG L
26	HSAVEP	56	*LBL 01
27	X=0?	57	CLX
28	RTN	58	LASTX
29	*LBL 03	59	GTO 03
30	HEPDIRX	60	END

**"HRESZFL" – Resize File**

ALPHA: H:File Name

Written by Steen Petersen, this routine was contributed in the HEPAX Manual. You can use it to resize Data or ASCII files; either upsize it so more records can be added to it, or downsize it to optimize the space used in cases that not all records are used (i.e. removing blank records at the end of the file). For example: Removing blank records of an ASCII file is easy to do:

HFLSIZE, HASROOM, 7, / , - , "HRESZFL"

This routine is also used in the ASCII File editor **HASED** at the beginning and end of the editing session. Note that ASCII Files in H:RAM *cannot be larger than 557 records*.

## Program Listing.-

The strategy of the routine is to copy all records into an auxiliary file ("\$\$") with the new size, removing the original afterwards and renaming the auxiliary file with the same name. This means sufficient available space must exist in the FileSystem to hold both files during the execution.

<b>01*LBL "HRESZFL"</b>	32 SF 25	63 HAPPCR	94 FC? 25
02 STO 00	33*LBL 03	64 FC?C 25	95 GTO 16
03 ABS	34 CLA	65 GTO 05	96 "\$"
04 ,	35 ARCL 01	66 GTO 16	97 HRCLPTA
05 HSEKPTA	36 ARCL 02	67*LBL 01	98 RDN
06 RDN	37 HRCLPTA	68 RCL 00	99 HSAVEX
07 HFLSIZE	38 "\$,"	69 X<0?	100 FS? 25
08 ASTO 01	39 HARCLRC	70 GTO 07	101 GTO 09
09 ASHF	40 FC? 25	71 SF 25	102 GTO 16
10 ASTO 02	41 GTO 16	72 HSEKPT	103*LBL 15
11 SF 25	42 HRCLPTA	73 FC? 25	104 "H:FL SIZE ERR"
12 HASROOM	43 ATOX	74 GTO 07	105 AVIEW
13 FC?C 25	44 ATOX	75 .	106 GTO 10
14 GTO 01	45 FC? 00	76*LBL 08	107*LBL 16
15 CLA	46 HAPPREC	77 HGETX	108 CLA
16 HAPPCR	47 FS? 00	78 X#Y?	109 ARCL 01
17 RCL 00	48 HAPPCR	79 GTO 15	110 ARCL 02
18 X<0?	49 FC? 25	80 FS? 25	111 HPURFL
19 GTO 02	50 GTO 05	81 GTO 08	112 "\$,"
20 RDN	51 CF 00	82 HSEKPT	113 ARCL 01
21 7	52 FS? 17	83*LBL 07	114 ARCL 02
22 /	53 SF 00	84 "\$"	115 HRENAME
23 INT	54 GTO 03	85 RCL 00	116*LBL 10
24 -	55*LBL 05	86 HCRFLD	117 CLA
25 X>Y?	56 -1	87 SF 25	118 ARCL 01
26 GTO 15	57 AROT	88*LBL 09	119 ARCL 02
27*LBL 02	58 ATOX	89 CLA	120 ,
28 "\$"	59 SF 25	90 ARCL 01	121 HSEKPT
29 RCL 00	60 FC? 00	91 ARCL 02	122 RCL 00
30 HCRFLAS	61 HAPPREC	92 HRCLPTA	123 CF 00
31 CF 00	62 FS? 00	93 HGETX	124 END

**PCOPY** – Copy program from ROM.

ALPHA: Program Name

The function that HP left out of the X-Functions ROM – likely due to space restrictions – is finally here. In fact, this is just the same **COPY** code preceded and trailed by a few more instructions to manage the FOCAL pointer so that the execution returns to the calling program. Unfortunately COPY ends with a call to [NFRPU] so its code cannot be used in a subroutine and therefore needs to be integrally copied to the module, oh well, such is life.

The price we pay for this trick is *losing the contents of the T-register*, so bear that in mind.

099	"Y"	
010	"P"	<u>Program Name in ALPHA</u>
00F	"O"	
003	"C"	
010	"P"	HP Co. - Ángel Martin
188	SETF 11	default: ROM
0CC	?FSET 10	pointer in ROM?
017	JC +02	yes, skip over
184	CLRF 11	no, flag it
141	?NC XQ	Retrieves the PC
044	->2950	[GETPC]
0AE	A<>C ALL	
028	WRIT 0(T)	destroys T
1B5	?NC XQ	Read Name from Alpha
114	->4560	[READNM]

.....  
Original COPY here, minus call to [NFRPU]  
.....

18C	?FSET 11	were we in RAM?
013	JNC +02	yes, skip
0C8	SETF 10	and flag ROM
188	SETF 11	reset defaults
046	C=0 S&X	
270	RAMSLCT	
038	READATA	get T register
10E	A=C ALL	
0DD	?NC XQ	
08C	->2337	[PUTPC]
31D	?NC GO	
002	->00C7	[NFRKB]

Note that **PCOPY** works on any user program residing in ROM, not only with Program Files in the H:RAM FileSystem.

This function is used in the routines **HUPDP** and **HGETFL/HSAVEFL** to transfer program files between X-Mem and H:RAM under program control.

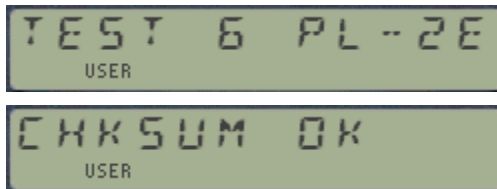


**ROMTEST** – ROM Checksum  
**ROMCHKX** – ROM Checksum

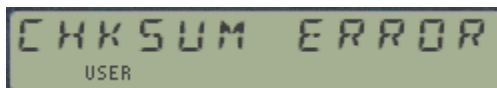
X: Page#  
X: XROM id#

These two functions perform the same job, namely calculating the ROM checksum that goes into the last position of each ROM and comparing it with the current value. The difference is that **ROMTEST** expects the page number in the X-register and **ROMCHKX** uses the XROM id# instead. Messaging is also slightly different, as can be seen in the examples below:

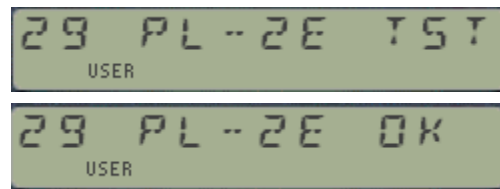
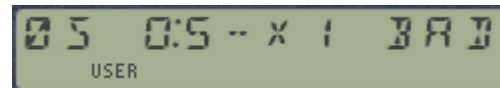
**6, ROMTEST:**



Or in case of a wrong checksum result:



**29, ROMCHKX:**


**RAMTEST** – Checking RAM Pages

X: Page #

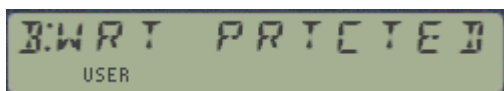
The analogous function for RAM checking is RAMTEST. It expects the page number in X and does an exhaustive test to each location within the tested page – verifying that the written values are correct.



When read/write errors are found the function reports the error below:



If the RAM page is protected (via RAMTOG) the function returns the warning message below:



## HP-IL File 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.

<b>FSIZE</b>	<b>HPIL Drive File Size</b>	FileName in ALPHA	<i>Assembler3 ROM</i>
<b>READF</b>	<b>Read Data File</b>	IL FName, XM FName	<i>Raymond del Tondo</i>
<b>WRTDF</b>	<b>Write Data File</b>	XM FName, IL FName	<i>Raymond del Tondo</i>

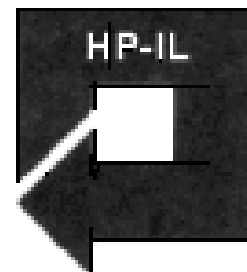
- **FSIZE** Returns to X the length in registers of the (primary) mass storage file which name is specified in Alpha. If no HP-IL is present on the system the error message "NO HPIL" will be shown.
- **READF** and **WRTDF** are used to read and write individual DATA files between the IL Drive and XMEM. 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	<b>LBL "GETDF"</b>	08	<b>LBL "SAVEDF"</b>
02	<b>FSIZE</b>	09	FLSIZE
03	<b>ASWAP</b>	10	<b>ASWAP</b>
04	CRFLD	11	CREATE
05	<b>ASWAP</b>	12	<b>ASWAP</b>
06	<b>READF</b>	13	<b>WRTDF</b>
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 among other sources.



## *Proposed Standard HP41 Machine language Bar Code*

---

Written by Steen Petersen, the following two HP-41 machine language routines will enable you to make bar code from HP-41 machine language (M-Code). These bar codes will be type 0, which is unused by the normal HP-41 system.

### **MCPR** - M-Code Bar Code PRINT

**MCPR** works like **BCP** in the HP-41 Plotter module, and creates a pattern in the ALPHA register to make bar codes from . You will need the Plotter module to print the bar codes on a HP7470 or on a HP82162.

The format of the bar codes is as follows:

- Bit 0 -> 7 contains the checksum, which is the sum of all the bytes with wrap-around carry, not a running checksum.
- Bit 8 -> 11 are all set to 0, the type number of M-Code bar code.
- Bit 12 -> 15 contains the row number minus one, mod 16 (example: row 1 = 0).
- Bit 16 -> 15+10\*N contains the information of N M-Code instructions, each occupying 10 bits.
- Bit 16+10+N is *the* end indicator. If it is one, this row is the last row; if it is zero, you will be prompted for more rows. After this row there will be between 1 and 7 bits to fill up the last byte in the row.

The final bar code looks as follows:

		Type Seq.	N M-code	End		
		Checksum =0	NO. instructions	bitFiller		

Instructions for use:

Input:

- Y-reg.: 000000|aaaa|bbbb Where **aaaa-bbbb** indicate the interval from which to make bar code.
- X-reg.: ccc,dd, where **ccc** is the row number, and **dd** is the number of instructions in each row. If dd=00 then 11 will be used (maximum length)

Note that it is not possible to have a row number higher than 999, giving a maximum of 8k 10-bits words of bar code in one sequence.

Output:

- Z-reg.: -ee,00 ; Where ee is *the* number of bytes in ALPHA register. Together with the minus sign, this is necessary to make bar codes on a HP82162 (LBL 01 , MCPR, RCI Z, BCO, RDN, X#0?, GTO 01 , RTN).
- Y-reg.: 000000aaaabbbb Unchanged.
- X-reg. : ccc+1,dd or 0 Increment to *the* next row if not done, else X-reg. is cleared.
- L-reg. : 000000ffffgggg ffff is the number of the first instruction in the row (if the instructions are numbered sequentially from 0000 and onward ), and gggg is the number of the last instruction.

If used together with a HP7470 Plotter, the MCPR function must be followed by the BC function in the Plotter Module.

Error messages:

DATA ERROR	if aaaa>bbbb, aaaa+2000<bbbb or dd>11.
OUT OF RANGE	if ccc > 999.

### MCSCAN - M-Code Bar Code SCAN

**MCSCAN** reads the bar codes made by MCPR. You will of course need a wand. It works as follows:

Input :

- X-reg.: 0000000000aaaa; Where aaaa is the first address you want to read bar codes to.

Output:

- X-reg.: 000000|aaaa|bbbb; Where aaaa is the start address and bbbb is the address of the last instruction (see SST) plus one.
- L-reg. : 0000000000|aaaa; True LASTX.

If you want to read several programs immediately after each other, you can just repeat using MCSCAN without thinking of input expect for the first program.

During the execution of MCSCAN the following keys are active:

- **ON** Turns the HP~41 off,
- **←** Terminates the MCSCAN routine,
- **R/S** As above
- **SST** Asks for the next row (skips one row) if it is not possible to read the current row. The words skipped will not be cleared as it may be possible to read the missing rows later. The number of instructions skipped will be the number of instructions in the last read row. If the first row is skipped, the jump will be calculated after the first possible row has been read. The X-reg. will be updated after each read row and after SST (if row 1 not skipped).

Once you have entered the MCSCAN routine, some routines in the wand-ROII will be used, setting the time-out period to approx. 7 min. at normal speed. You will be prompted as usual: "W: RDY nnn" to scan row nnn. If the MCSCAN routine meets an end-bit set to one, the routine will terminate.

Error messages:

<b>NO WAND</b>	if wand no present,
<b>W: CHKSUM ERR</b>	if error in checksum, try again.
<b>W: TYPE ERR</b>	if you are trying to read non M-Code bar code.
<b>W: SEQ ERR</b>	if you are trying to read in a wrong sequence
(this message will not occur if you read exactly 16 rows wrong).	

Both MCPR and MCSCAN will be in the BOOT ROM EPROMI set, expected to be completed soon. MCPR and MCSCAN are released for non-commercial use only, but the type zero bar code described under MCPR is of course free to anyone.

(c) Copyright 1985 Steen Petersen, PPC-Denmark

## *The sequel: HEPAX Disassembler ROM*

---

A derivative of this project is the Hepax Disassembler ROM. As you can expect by its name, it contains the **DISASM** function – extracted from the fourth bank in the original module – but it also adds a few functions to complement the disassembling functionality, such as the single-step execution.

Here is the list of functions available in this module:

1. **DISASM** - Hepax Disassembler
2. **"DISSST** - Single-step DISASM
3. **CODE / DECODEYX** Used in DISSST program
4. **HPROMPT** - Used in DISSST program
  
5. **"CLHM** - Clears all files in the Hepax File System
6. **"HRESZFL** - Hepax File Resize
7. **"?JUMP** - Gets MCODE for jump instructions
  
8. **RLSRAM** - Releases page from RAM Hepax chain
9. **HEPCHN?** - Shows configured RAM Hepax chain
10. **HEPCHN** - Re-configures RAM Hepax chain

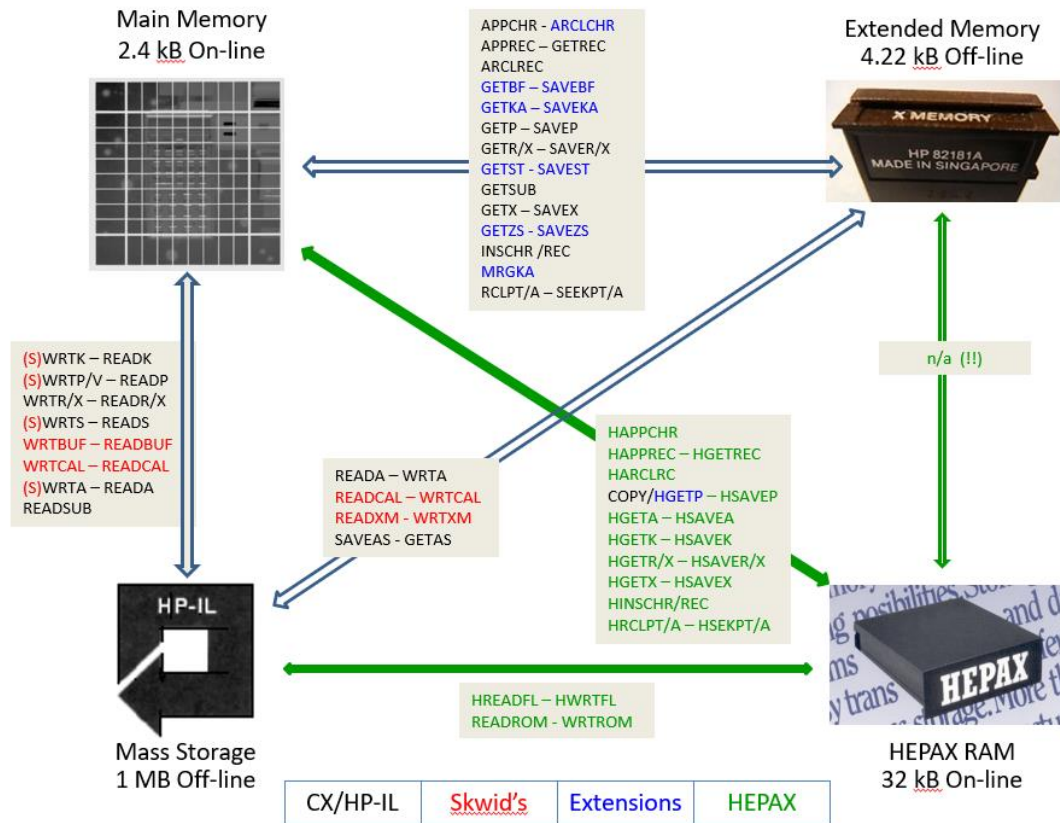
You have no doubt recognized **DISSST**, **HRESZFL** and **?JUMP**, from the HEPAX Manual. **CLHM** is a small example of utilization, which could be enhanced with **HFLTYP** to only delete those files of a given type, say DATA for instance.

Because of the inclusion of the auxiliary functions **CODE**, **DECODEYX** and **HPROMPT**, the program DISSST from this module is self-contained and can be executed without the HEPAX module plugged in the calculator. Note however that the other FOCAL programs included require the HEPAX module also to be present.

In summary, the HEPAX Disassembler is partially independent from the HEPAX itself – either the original or the enhanced revision 1G. Both can coexist being plugged in simultaneously although there is a large overlap between them - the **DISASM** functionality being the most obvious one. Note also that even if it's not required for the basic FileSystem operations, it's very recommendable to have the Library#4 plugged in at all times. The extended CATalogs will fail if not present!

## HEPAX RAM vs. Other Memory Types

The picture below shows the analogies and correspondence between HEPAX RAM and other types available in the 41 System. It includes all functions and sub-functions available in the PowerCL and other advanced modules.



## HEPAX File Types vs. Other Systems

File Type	HEPAX	X-Mem	HP-IL
Program	HSAVEP - HGETP (COPY)	SAVEP - GETP	WRTP/V - READP
Data	HSAYER/X - HGETR/X	SAVEREG/X - GETRG/X	WRTR/X - READR/X
ASCII	HAPPREC - HGETREC	APPREC - GETREC	SAVEAS - GETAS
Matrix	HSAVEM, HGETM	MATDIM, et al.	n/a
Key Assignments	HSAVEK - HGETK	SAVEKA - GETKA	WRTK - READK
Buffer	HSAVEB, HGETB	SAVEBF - GETBF	WRTBUB - READBUF
Complex Stack	n/a	SAVEZS - GETZS	n/a
Status Regs	n/a	SAVEST - GETST	WRTS - READS
16C Buffer	n/a	SAVE16C - GET16C	n/a
ALL	HSAVEA - HGETA	n/a	WRTA - READA
X-Mem	n/a	n/a	WRTXM - READXM
CALC	n/a	n/a	WRTCAL - READCAL
ROM	COPYROM	n/a	WRTROM - READROM



