HP-41 XMEM-TWIN MODULE

X-Register Management and Stack Swap Functions
With Block & Sorting Applications
Revision 3B

Written and programmed by Ángel Martin
April 8, 2023

This compilation revision 1.2.5

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

Acknowledgments.- This module is a derivative of the "CL Expanded Registers" project. A 41-CL is not required.

Everlasting thanks to the original developers of the HEPAX and CCD Modules – real landmarks 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.

Module Main Function Table and Descriptions.

#	Function	Description	Dependency	Type	Author
0	**-XMEM TWIN**	*Lib#4 Check & Splash*	Lib#4	MCODE	*Nelson F. Crowle*
1	**YF$ _**	Sub-function Launcher by Name	Lib#4	MCODE	*Ángel Martin*
2	**YF# _ _ _**	Sub-function Launcher by index	Lib#4	MCODE	*Ángel Martin*
3	**A<>XRG _ _ _**	Swap Alpha and X-Regs	Lib#4	MCODE	*Ángel Martin*
4	**CLXRG**	Clear All Extended Regs	Lib#4	MCODE	*Ángel Martin*
5	**CLXRGX**	Clear X-Regs by X	Lib#4	MCODE	*Ángel Martin*
6	**CPYBNK _ "_:_**	Copy Banked Block	Lib#4	MCODE	*Ángel Martin*
7	**ST<>XRG _ _ _**	Swap Stack and X-Regs	Lib#4	MCODE	*Ángel Martin*
8	**STKSWP _**	*Stack Swap Launcher*	Lib#4	MCODE	*Ángel Martin*
9	**XFINDX**	Searches for value in X in X-Regs	Lib#4	MCODE	*Ángel Martin*
10	**XRGMOV**	Move Expended Regs Blocs	Lib#4	MCODE	*Ángel Martin*
11	**XRGSWP**	Swap Expanded Regs Blocks	Lib#4	MCODE	*Ángel Martin*
12	**-X-REGS FNS**	Section Header	n/a	MCODE	*Ángel Martin*
13	**XARC _ _ _**	X-Regs ARCL	Lib#5	MCODE	*Ángel Martin*
14	**XAST _ _ _**	X-Regs ASTO	Lib#6	MCODE	*Ángel Martin*
15	**XDSE _ _ _**	Expanded Reg DES	Lib#4	MCODE	*Ángel Martin*
16	**XISG _ _ _**	Expanded Reg ISG	Lib#4	MCODE	*Ángel Martin*
17	**XRCL _ _**	Extended Recall	Lib#4	MCODE	*Ángel Martin*
18	**XRC+ _ _ _**	XRCL Addition	Lib#4	MCODE	*Ángel Martin*
19	**XRC- _ _ _**	XRCL Subtract	Lib#4	MCODE	*Ángel Martin*
20	**XRC* _ _ _**	XRCL Multiply	Lib#4	MCODE	*Ángel Martin*
21	**XRC/ _ _ _**	XRCL Divide	Lib#4	MCODE	*Ángel Martin*
22	**XSTO _ _**	Extended Store	Lib#4	MCODE	*Ángel Martin*
23	**XST+ _ _ _**	XSTO Addition	Lib#4	MCODE	*Ángel Martin*
24	**XST- _ _ _**	XSTO Subtract	Lib#4	MCODE	*Ángel Martin*
25	**XST* _ _ _**	XSTO Multiply	Lib#4	MCODE	*Ángel Martin*
26	**XST/ _ _ _**	XSTO Divide	Lib#4	MCODE	*Ángel Martin*
27	**XVEW _ _ _**	Extended View	Lib#4	MCODE	*Ángel Martin*
28	**XX<> _ _**	Extended Exchange	Lib#4	MCODE	*Ángel Martin*
29	**-X-REGS APPS**	Section Header	n/a	MCODE	*Ángel Martin*
30	**WORKFL**	Working File Name	Lib#4	MCODE	*Sebastian Toelg*
31	**"XDUMP**	Dumps Standard Regs into X-	None	FOCAL	*Ángel Martin*
32	**"XINPT**	Inputs X-Regs	AMC_OSX	FOCAL	*Ángel Martin*
33	**"XOUT**	Outputs X-Regs	AMC_OSX	FOCAL	*Ángel Martin*
34	**"XRAN**	Enters Random values	AMC_OSX	FOCAL	*Ángel Martin*
35	**"XSHFT**	Selective Std, Reg copy	none	FOCAL	*Ángel Martin*
36	**"XSORT**	Sorts X-Regs	none	FOCAL	*JM Baillard*
37	**“XS1**	PPC Stack Sort	none	FOCAL	*PPC Members*
38	**“XS2**	PPC Small Set Sort	none	FOCAL	*PPC Members*
39	**“XS3**	PPC Large Set Sort	none	FOCAL	*PPC Members*
40	**-FREGS FNX**	Section Header	n/a	MCODE	n/a
41	**FARC _ _ _**	File-Regs ARCL	Lib#5	MCODE	*Ángel Martin*
42	**FAST _ _ _**	File-Regs ASTO	Lib#6	MCODE	*Ángel Martin*
43	**FDSE _ _ _**	File Reg DES	Lib#4	MCODE	*Ángel Martin*
44	**FISG _ _ _**	File Reg ISG	Lib#4	MCODE	*Ángel Martin*
45	**FRCL _ _**	File Reg Recall	Lib#4	MCODE	*Ángel Martin*
46	**FRC+ _ _ _**	FRCL Addition	Lib#4	MCODE	*Ángel Martin*
47	**FRC- _ _ _**	FRCL Subtract	Lib#4	MCODE	*Ángel Martin*
48	**FRC* _ _ _**	FRCL Multiply	Lib#4	MCODE	*Ángel Martin*
49	**FRC/ _ _ _**	FRCL Divide	Lib#4	MCODE	*Ángel Martin*
50	**FSTO _ _**	File Reg Store	Lib#4	MCODE	*Ángel Martin*
51	**FST+ _ _ _**	FSTO Addition	Lib#4	MCODE	*Ángel Martin*
52	**FST- _ _ _**	FSTO Subtract	Lib#4	MCODE	*Ángel Martin*

53	**FST* _ _ _**	FSTO Multiply	Lib#4	MCODE	*Ángel Martin*
54	**FST/ _ _ _**	FSTO Divide	Lib#4	MCODE	*Ángel Martin*
55	**FVEW _ _ _**	File Reg View	Lib#4	MCODE	*Ángel Martin*
56	**FX<> _ _**	File Reg Exchange	Lib#4	MCODE	*Ángel Martin*
57	**-FREGS APPS**	Section Header	n/a	MCODE	n/a
58	**POSDF**	Position in Data File	Lib#4	MCODE	*Ángel Martin*
59	**"DFED**	Data File Editor	AMC_OSX	FOCAL	*Ángel Martin*
60	**"FLRAN**	Randomizes Data File	AMC_OSX	FOCAL	*Ángel Martin*
61	**"FLSHFT**	Copies Data Regs to File	none	FOCAL	*Ángel Martin*
62	**"FLSORT**	Bubble-Sorts Data File	none	FOCAL	*Ángel Martin*
63	**"FLDUMP**	Dumps all Data Regs to File	none	FOCAL	*Ángel Martin*

This module also includes a set of sub-functions arranged in an Auxiliary FAT, as follows:

0	**-STK SWAPS**	Section Header	Lib#4	MCODE	*Ángel Martin*
1	**a<> _ _**	Swap a and register	Lib#4	MCODE	*Ángel Martin*
2	**b<> _ _**	Swap b and Register	Lib#4	MCODE	*Ángel Martin*
3	**c<> _ _**	Swap c and register	Lib#4	MCODE	*Ángel Martin*
4	**d<> _ _**	Swap d and Register	Lib#4	MCODE	*Ángel Martin*
5	**e<> _ _**	Swap e and register	Lib#4	MCODE	*Ángel Martin*
6	**}<> _ _**	Swap }- and Register	Lib#4	MCODE	*Ángel Martin*
7	**L<> _ _**	Swap L and register	Lib#4	MCODE	*Ángel Martin*
8	**M<> _ _**	Swap M and Register	Lib#4	MCODE	*Ángel Martin*
9	**N<> _ _**	Swap N and register	Lib#4	MCODE	*Ángel Martin*
10	**O<> _ _**	Swap O and Register	Lib#4	MCODE	*Ángel Martin*
11	**P<> _ _**	Swap P and register	Lib#4	MCODE	*Ángel Martin*
12	**Q<> _ _**	Swap Q and Register	Lib#4	MCODE	*Ángel Martin*
13	**T<> _ _**	Swap T and register	Lib#4	MCODE	*Ángel Martin*
14	**Y<> _ _**	Swap Y and Register	Lib#4	MCODE	*Ángel Martin*
15	**Z<> _ _**	Swap Z and register	Lib#4	MCODE	*Ángel Martin*
16	**ANUMDL**	Gets number from ALPHA	Lib#4	MCODE	*HP Co.*
17	**ASWP>**	Swaps ALPHA around '>"	Lib#4	MCODE	*W&W GmbH*
18	**CLEM**	Clear X-Mem	Lib#4	MCODE	*Hakan Thörngren*
19	**D>H**	Decimal to Hex	Lib#4	MCODE	*William Graham*
20	**H>D**	Hex to Decimal	Lib#4	MCODE	*William Graham*
21	**LKAOFF**	Suspend Local KA	Lib#4	MCODE	*Ross Cooling*
22	**LKAON**	Reestablish Local KA	Lib#4	MCODE	*HP Co.*
23	**RECADR**	Record Address	Lib#4	MCODE	*Ángel Martin*
24	**STVIEW**	**Stack Viewer**	Lib#4	MCODE	*Ángel Martin*
25	**ULAM**	**Ulam's Conjecture**	Lib#4	MCODE	*Ángel Martin*
26	**Y/N?**	Yes/No Prompt	Lib#4	MCODE	*Ángel Martin*
27	**CAT+**	Sub-function Catalog	Lib#4	MCODE	*Ángel Martin*

Optical isomer or Doppelgänger?

Introduction – Unleashing the Extended Registers

If you ever had questions about the hp-41 X-Mem design and implementation, perhaps wishing that HP had done something *different* (and this is not implying that the actual model is a bad one), then you may enjoy the chance to travel "the path not taken", playing around with this new module.

Meet the XMem-(e)TWIN module, where the "e" in this case stands for both electronic and evil - Why evil, you ask? Because those void memory segments in-between X-Mem modules certainly deserve such a moniker; a real struggle to manage in the MCODE and a waste of available space (what were they thinking?) -- but let's not digress...

The XMEM_TWIN module provides a complete set of functions targeted to the X-Registers that make the Extended memory area. Within X-Mem there are 606 X-Registers that can now be accessed individually using X-versions of (A)RCL, (A)STO, X<>, ISG, DSE, VIEW, etc., fully supporting direct and indirect addressing, for the three ranges of data sources: Stack Registers, "regular" Data registers, and X-Registers.

Obviously once you get into this path there's no use for any of the "official" X-Mem functions, so this is an either/or choice. But it's totally "reversible" (for the lack of a better word), simply use CLXM to start over from a clean slate using the orthodox way. Remember: X-Mem files will not survive the use of the X-Regs functions!

New in Version 3A

By popular demand, the module also includes a section that provides control of the individual registers *within an X-Mem Data File*, in a manner fully compatible with the X-Mem directory contents: all file headers and X-Mem control registers are respected, thus no risk of data loss. The "in-file" F-Regs functions are only restricted by the data file size, which can be as large as 600 registers if this is the only file in the directory.

A reasonable amount of testing has been done but this is never a guarantee for bug-free software (remember those memory voids, always lurking in the dark?) - so be careful and DO NOT use it without a data backup. Use it at your own risk, as they say. Hope you enjoy playing with this evil twin, and as usual feedback is always welcome.

Module Dependencies.

The XMEM Twin is a Library4-aware module; therefore, it expects the Library#4 revision R4 to be present on the system. The module will check for it upon the calculator ON event, showing an error message if not found. This will abort the polling points sequence for all other modules plugged at higher position in the bus. *Do not attempt to run the programs or functions within the module without the Library#4 plugged in.*

The AMC_OS/X Module is also required to run some of the FOCAL programs from APPS sections. This module provides advanced OS extensions and therefore it's recommended to have it always plugged in the machine – any real power user can't live without it.

Note: The X-Registers module requires the Library#4 revision R4 or higher plugged in.

I. Managing the Global X-Mem Extended registers

This first section covers the individual access of the extended registers included in the three X-MEM blocks. You'll be able to store, recall, view, exchange, and perform ISG/DSE operations on 606 of those registers (from 0 to 605) as if they were standard data registers within main memory. Note the presence of the arithmetic operations as well.

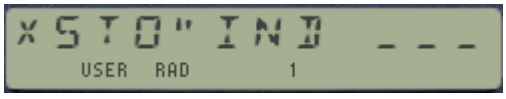
Extended Regs	Store	Recall	Other	X-Blocks
X-Register	**XSTO _ _ _**	**XRCL _ _ _**	**XX<> _ _ _**	**CLXRG**
From 0 to 605.	**XST+ _ _ _**	**XRC+ _ _ _**	**XVEW _ _ _**	**CLXRGX**
	XST- _ _ _	**XRC- _ _ _**	**XDSE _ _ _**	**XRGMOV**
	XST* _ _ _	**XRC* _ _ _**	**XISG _ _ _**	**XRGSWP**
	XST/ _ _ _	**XRC/ _ _ _**	**XFINDX**	**ST<>XRG _ _ _**
ALPHA	**XAST _ _ _**	**XARC _ _ _**		**A<>XRG _ _ _**

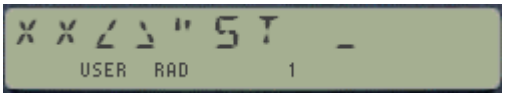
Besides the direct access, you also have the INDirect addressing capabilities implemented on the expanded registers; the sixteen Stack registers (including synthetic regs {M-e}); and all the standard-Registers - a hybrid mode, unique to this implementation.

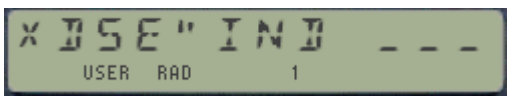
Most of the functions will prompt for the parameters to use. The initial prompt is a three-field underscore for the X-register indeed. Pressing [SHIFT] changes it to IND three-digit fields for another X-register to be used as indirect. Pressing the [RADIX] key changes to the IND ST _ prompt, where you'll enter the register mnemonic, from T to e (all sixteen are available). Pressing the radix key again changes to a IND RG_ _ prompt where you can enter a standard register number to use as indirect address. Repeat pressings of the radix key act as a toggle between those two. There's also provision for direct stack and standard register arguments – even if those can be redundant in practice, being exactly the same as the original ones.

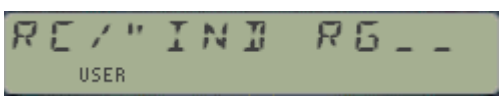
Once you complete the entry adding the register number the action is performed in RUN mode, or two program lines are entered in program mode – automatically selecting the appropriate parameter depending on the direct or indirect types. This is automatically done so you needn't (and shouldn't) edit the value entered in the program's second line at all – which will be properly interpreted in a running program.

You can move between the functions while the prompts are up; not only to select the math operation but also to change the main function amongst the group. So for instance during the **XRCL** _ _ _ main prompt pressing the [SST] key will trigger the **XX<>** function, or pressing [STO] will invoke the **XSTO** function instead. Also *you can revert to the original mainframe functions* pressing the corresponding key of the function in the prompt, for instance here pressing [RCL] will trigger the original RCL _ _



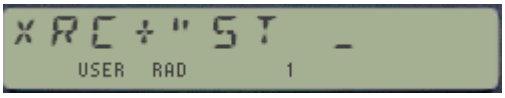






The functions will not allow you to enter any value greater than 605 either as direct index or indirect index – not even when entering them in a program line. Attempting to enter larger values will trigger a "NONEXISTENT" error message. However, that check is not made for IND_RG combinations, as there's no telling at that point about how many standard registers will be available at the execution stage.

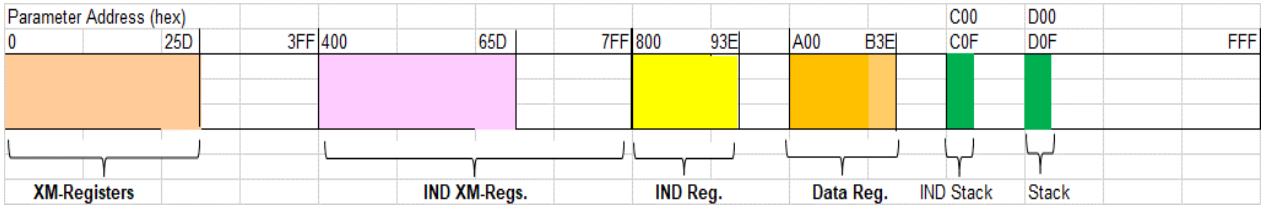
The usage of standard stack and data registers is not only more convenient from the usability standpoint, but also it enables the RCL math on these registers via the **XRCL** function:



Although possible, it is however not meant to be used in a program because of the obviously higher byte count. That's why when used in an editing program the direct stack and data register X-functions *revert automatically to the native STO/RCL functions instead*, which has the additional benefit of a clearer representation by the OS as merged lines. *This includes the STO_Math functions as well.*

The table and chart below show all possible combinations for program editing:

Func.	Y-Reg nnn	adds 0x400 IND Y-reg nnn	adds 0xA00 Data Reg nn	adds 0x800 IND Data Reg nn	adds 0xD00 STK nn	adds 0xC00 IND STK nn
XSTO	nnn	nnn+1024	**STO nn**	nn+2048	**STO ST(nn)**	nn+3072
XST+	nnn	nnn+1024	**ST+ nn**	nn+2048	**ST+ ST(nn)**	nn+3072
XST-	nnn	nnn+1024	**ST- nn**	nn+2048	**ST- ST(nn)**	nn+3072
XST*	nnn	nnn+1024	**ST* nn**	nn+2048	**ST* ST(nn)**	nn+3072
XST/	nnn	nnn+1024	**ST/ nn**	nn+2048	**ST/ ST(nn)**	nn+3072
XRCL	nnn	nnn+1024	**RCL nn**	nn+2048	**RCL ST(nn)**	nn+3072
XRC+	nnn	nnn+1024	nn+2560	nn+2048	nn+3328	nn+3072
XRC-	nnn	nnn+1024	nn+2560	nn+2048	nn+3328	(nn+3072
XRC*	nnn	nnn+1024	nn+2560	nn+2048	nn+3328	nn+3072
XRC/	nnn	nnn+1024	nn+2560	nn+2048	nn+3328	nn+3072
XDSE	nnn	nnn+1024	**DSE nn**	nn+2048	**DSE ST(nn)**	nn+3072
XISG	nnn	nnn+1024	**ISG nn**	nn+2048	**ISG ST(nn)**	nn+3072
XX<>	nnn	nnn+1024	**X<> nn**	nn+2048	**X<> ST(nn)**	nn+3072
XVEW	nnn	nnn+1024	**VIEW nn**	nn+2048	**VIEW ST(nn)**	nn+3072
XARC	nnn	nmn+1024	**ARCL nn**	nn+2048	**ARCL ST(nn)**	nn+3072
XYAST	nnn	mnn+1024	**ASTO nn**	nn+2048	**ASTO ST(nn)**	nn+3072

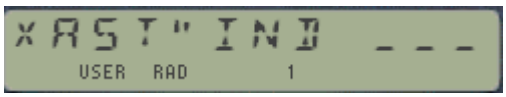


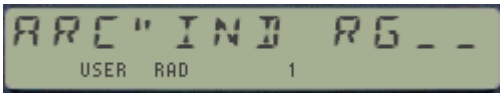
Note that the U/I won't allow entering Data Register indexes above 99. If values 318>nn>99 are needed you should edit manually the second non-merged line, using the same rule as shown above. This will only apply to the XRCL_Math functions, as all others have a native OS equivalent.

XM-REG Data_Reg nn = XM-REG as first line plus (nn+2560) as second line.
XM-REG IND Data_Reg nn = XM_REG as first line plus (nn+2048) as second line.

Storing and Recalling ALPHA Data

The extended functions **XAST** and **XARC** provide the means to store and recall ALPHA data directly in the expanded registers area. Like their numeric counterparts, they support direct, INDirect, stack and standard registers indexes for a complete palette of options at your disposal. You can access these directly from the XSTO/XRCL prompts by pressing the ALPHA key at any time.





Going above 604

The U/I will throw a "NONEXISTENT" error message when you type a numer above 605 at the manual prompt, or when the second line of the non-merged instruction has a value larger than 604.

Deleting, Moving and Swapping Expanded Registers.

Think of the following functions as analogous to the X-Functions extensions on the original function set of the calculator, only applied to the extended memory area instead.

- The function **CLXRG** will delete all the 605 expanded registers.
- Additionally with **CLXRGX** you can selectively delete a defined block of extended registers as defined by its control word (in X) "bbb.eeennn", The bbb digits are the base address of the source expanded register block, in the range R0 through R605. The eee digits are the base address destination expanded register block, again in the range R0 through R605. nnn is the incremental step for the registers to delete. If nnn is zero a value of one used.
- **XRGMOV** and **XRGSWP** can be used to move or exchange a block of extended registers at once – either contiguous or in an increment pattern as provided by the control word sss.dddii in the X-register. Much the same as the X-Functions RGMOVE and RGSWAP - in case you wonder.

Other Block Operations.

- **A<>XRG** and **ST<>XRG** exchange a group of five extended registers with ALPHA (plus Q) or the Stack (T-L) respectively. The start register is to be entered at the prompt in manual mode, or expected to be in the X-Register when running a program. These functions do not allow INDirect indexing.

Moving around the Extended Registers Functions.

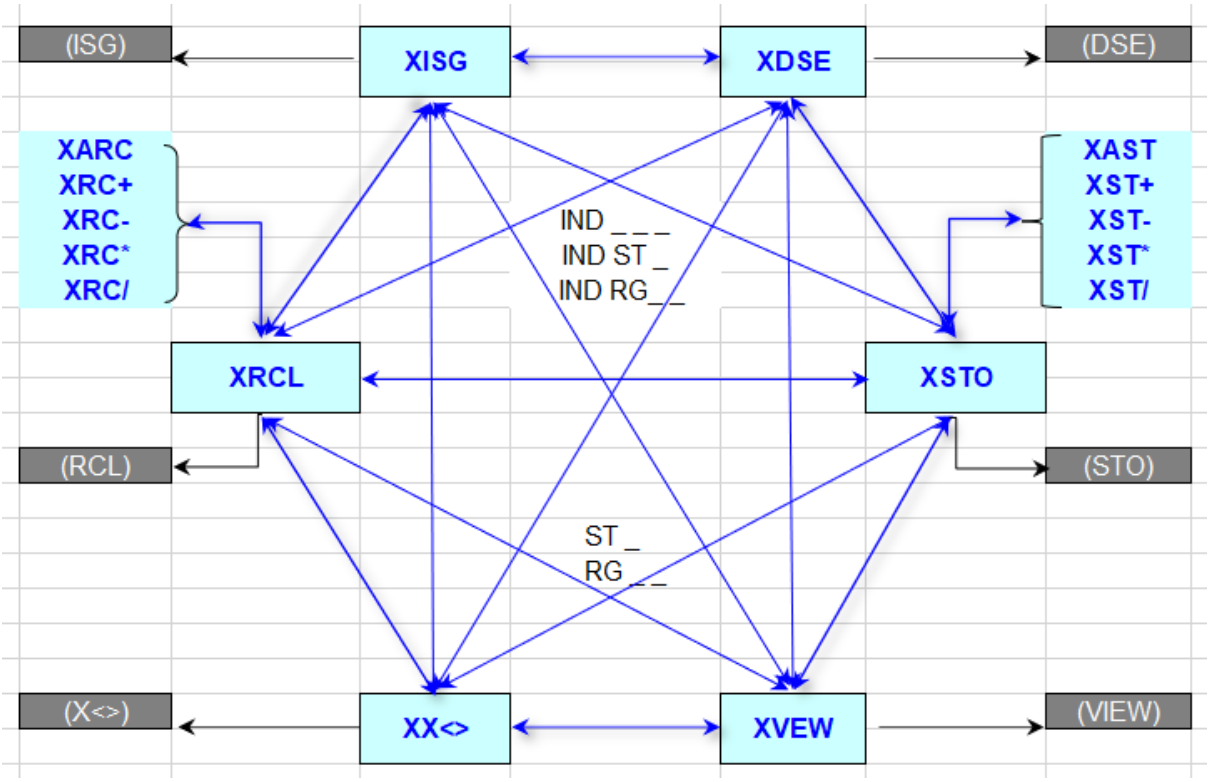


Figure 1.

Even if there isn't a dedicated launcher for these functions, navigation amongst them is as easy as intuitive. First off, assign one of the functions to its "natural" key, for instance **XSTO** to the STO key.

Then while the XSTO _ _ _ prompt is shown you can move about all the X-Reg functions by pressing the key for the corresponding action, i.e.

- RCL will toggle to XRCL _ _ _
- SST (for X<>) will launch XX<> _ _ _
- R/S (for VIEW) will launch XVEW _ _ _
- CHS (for ISG) will launch XISG _ _ _
- ALPHA will trigger XAST _ _ _
- The math keys will launch the corresponding math function, ie. XST+ _ _ _
- EEX (for DSE)will launch XDSE _ _ _
- SHIFT will add the IND _ _ prompt
- RADIX will add the ST _ prompt. All 16 status regs are selectable.
- RADIX again to toggle between ST _ and RG _ _. Choose any standard reg up to 99.
- STO again to exit to the native STO function (no way back!)
- Note that the DIRECT Stack/REG prompt is not strictly needed – that's the native function already. However, the RCL Math functions are useful and are available using this approach.

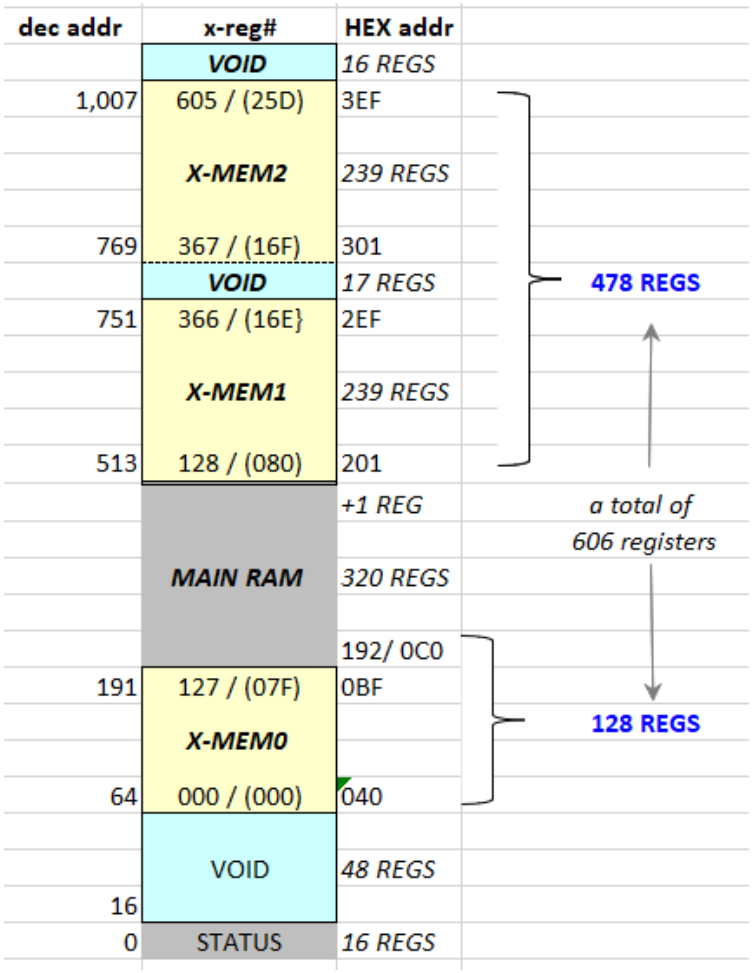
All the options above are available from within any of the 16 functions – regardless of which one you used to start the sequence. See the descriptions earlier in this manual for more details.

Mapping actual addresses to X-Registers.

The figure below depicts the memory arrangement of the three X-Mem modules and the corresponding mapping with the X-Register indexes. Note that:

- There are three void memory segments (in light blue), without any memory structure underneath.
- There are two reserved areas: The Status Registers and the Main RAM area, where all FOCAL programs and regular data registers reside.

Obviously not the memory voids, nor the reserved areas should be accessed by the X-Regs functions, if for different reasons, but specially not to mess with the current user programs and data registers in the Main RAM zone and Status regs.



Finally, the summary table below correlates the register indexes to the actual memory addresses. You can use other memory-peeking functions such as PEEKR and POKER to verify the operation, usually with the decimal addresses as parameters.

X-Reg Index#	Dec Offset	Dec address	HEX address
0 - 127	64	64 - 191	040 – 0BF
128 -366	385	513 - 751	201 – 2EF
367 - 605	402	769 – 1,007	301 – 3EF

Extra bonus: Finding the X-needle in the X-haystack.

For those times when you'd like to know if a certain value is stored in the X-data registers, the function **XFINDX** is available to do a cursory comparison looking for a match with the value in the X-register. All X-data registers are checked, starting with XR00 until XR605 – which could take a long time depending on where the match exists.

The function returns the number of the first X-data register found that contains the same value as the X-Register. If none is found, the function puts -1 in X to signify a no-match situation. The stack is lifted so the sought for value will be pushed to stack register Y upon completion.

Below there is a FOCAL routine that checks up to XR605, as well as an equivalent routine for the standard data registers - for comparison purposes. See the WARP_Corel manual for yet another routine to tackle this "where is Waldo" problem using other advanced functions.

```
01  LBL "XFNDX"
02  .605
03  X<>Y
04  LBL 00
05  XRCL IND Y (3074)
06  X=Y?
07  GTO 02
08  RDN
09  ISG X
10  GTO 00
11  CLX
12  -1
13  RTN
14  LBL 02
15  X<> Z
16  INT
17  END
```

```
01  LBL "FINDX"
02  SIZE?
03  E
04  –
05  E3
06  /
07  X<>Y
08  LBL 00
09  RCL IND Y
10  X=Y?
11  GTO 02
12  RDN
13  ISG Y
14  GTO 00
15  CLX
16  -1
17  RTN
18  LBL 02
19  X<> Z
20  INT
21  END
```

The possibilities of having an additional set of 606 registers available to your own programs are wide and deep. For starters you could permanently operate with a SIZE 000 and use all the 320 standard registers in main memory for User Code programs, key assignments and I/O buffers; so a few more bytes taken up by the parameter lines won't be a problem.

Converting Standard Programs

Having a complete function set ensures you can convert programs very easily, simply by replacing the standard functions with their expanded version. Even the ALPHA storage functions **XAST** and **XARC** are included, which can also use the expanded register range.

Then you have the benefit of a much larger set of registers (606 vs. a maximum of 319 without any program in RAM) available for your program, a sheer advantage to manage larger size cases of the problem you're trying to solve – from matrix operations to sorting data, to mention just a couple.

For example, with a few modifications the PPC ROM programs **S2** and **S3** can be used to sort more than 600 registers in a very efficient way. – with random data populating those registers it took about 20 seconds to sort 606 registers on TURBO 50 mode!

See below two simple routines I used to populate the registers and to view them. They expect the control word bbb.eee in X before you run them.

01	**LBL "XVIEW"**
02	LBL 00
03	"XR"
04	ARCLI
05	"|-: "
06	**XRCL IND X** (3075)
08	ARCLX
09	AVIEW
10	PSE
11	RDN
12	ISG X
13	GTO 00
14	END

01	**LBL "XRAN"**
02	RCL X
03	LBL 00
04	RNDM
05	**XSTO IND Y** (3074)
07	RDN
08	ISG X
09	GTO 00
10	RDN
11	END

Functions **ARCLI** and **RNDM** are available in the AMC_OX/X Module.

Note.- In case you're interested, the parameter lines used by these functions as non-merged, second line, correspond to the following:

1. The register index for direct access, from 0 to 605 [000 – 25D]
2. The indirect register index for IND from 1,024 to 1,629 [400 – 65D]
3. The hybrid standard register IND RG, from 2,048 to 2,367 [800 – 93F]
4. The direct standard registers, from 2560 to 2879 [A00 – B3F]
5. The indirect Stack register index, from 3,072 to 3,088 [C00 – C10]
6. The direct Stack registers index, from 3328 to 3343 [D00 – D10]

Obviously, there's a few gaps of unused values, like between 2,368 and 2,559 – but you shouldn't be concerned with this at all; after all the parameters are entered automatically by the functions (totally transparent to the user), and it takes the same number of bytes to use a 4-digit number, regardless of its value.

Note that the status register Q(9) is used internally by the function's MCODE, and therefore should not be used in your FOCAL programs as synthetic register when the expanded registers functions are also used.

A few Housekeeping Routines.

I've added a few more Input/output routines for convenience, including data movement between the standard and expanded registers zones These few new routines are briefly described below:

- **STVIEW** does a sequential enumeration of the stack values, {X,Y,Z,T,L}
- **RECADR** returns the absolute address of the record number in X, for the file whose name is in ALPHA. You can use the result with **PEEKR**/**POKER** to check your data.
- **XDUMP** copies *all standard registers* into the Y-Reg area, using the same indexes.

:ii

- **XSHFT** does a selective copy, using the control word bbb.eee:ii in X. Note that if this is negative then the direction of the copy is reversed, i.e. will go from the X-Area to the standard registers. In this case, make sure the SIZE is set so that there are enough standard registers to receive the X-data!
- **XINP** and **XOUT** are input/output routines to enter or visualize the values respectively. Require the control word bbb.eee in X.
- **XRAN** populates a block of X-Registers with random numbers, using **RNDM** from the OSX module (which takes its initial seed from the Time Module). Useful to test the sorting programs amongst other things.
- **XSORT** will do a descending data sort on the X-registers block defined by the control word bbb.eee in X. Note that *this is a very slow program*; use the PPC versions **XS2** and **XS3** for speed. Numeric values only!

See below the program listing for XRAN Abd XSORT

```
01  LBL "XSORT"
02  RUNNING
03  SIGN
04  LBL 01
05  LASTX
06  LASTX
07  XRCL IND ST L (3076)
08  LBL 02
09  XRCL IND ST Y (3074)
10  X>Y?
11  GTO 03
12  X<>Y
13  LASTX
14  +
15  LBL 03
16  RDN
17  ISG Y(2)
18  GTO 02
19  XX<> IND ST L (3076)
20  XSTO IND ST Z  (3073)
21  ISG L(4)
22  GTO 01
23  CLD
24  RTN
25  LBL "XRAN"
26  RCL X(3)
27  LBL 00
28  RNDM
29  XSTO IND ST Y (3074)
30  RDN
31  ISG X(3)
32  GTO 00
33  RDN
34  END
```

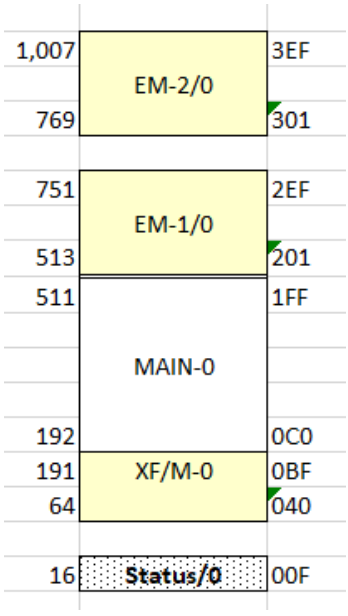
XDUMP and XSHFT program listing is shown below:

```
01 LBL "XDUMP"
02 SIZE?
03 E
04 –
05 E3
06 /
07 LBL "XSHFT"
08 CF 00
09 X<0?
10 SF 00
11 ABS
12 RUNNING
13 LBL 00
14 FC? 00
15 GTO 01
16 XRCL IND ST X (3075)
17 STO IND Y(2)
18 GTO 02
19 LBL 01
20 RCL IND X(3)
21 XSTO IND ST Y (3074)
22 LBL 02
23 RDN
24 ISG X(3)
25 GTO 00
26 CLD
27 END
```

Finally, XINP and XOUT program listing is below:

```
01 LBL "XOUT"
02 LBL 00
03 "XR"
04 ARCLI
05 >": "
06 XARC IND ST X (3075)
07 AVIEW
08 PSE
09 ISG X(3)
10 GTO 00
11 RTN
12 LBL "XINP"
13 "X"
14 ARCLI
15 >"="
16 XARCL IND ST X (3075}
17 >"?"
18 RDN
19 CF 22
20 PROMPT
21 FC?C 22
22 GTO 03
23 XSTO IND Y (3074)
24 RDN
25 LBL 03
26 ISG X(3)
27 GTO 01
28 END
```

The companion modules **XMPPC** and **XMMTRX** contain a large7 selection of Matrix and Registers applications programs taken from the PPC ROM collection. I have modified them to take advantage of the extended registers, replacing all operation from the standard registers – except the control parameters in the PPC routines, which are still using those. Make sure you have the PPC Manual or QRG handy when using these routines…



II. Managing the In-file X-Mem Extended registers

This first section covers the individual access of the extended registers included *in a given Data File*. You'll be able to store, recall, view, exchange, and perform ISG/DSE operations on those registers as if they were standard data registers within main memory. Note the presence of the arithmetic operations as well.

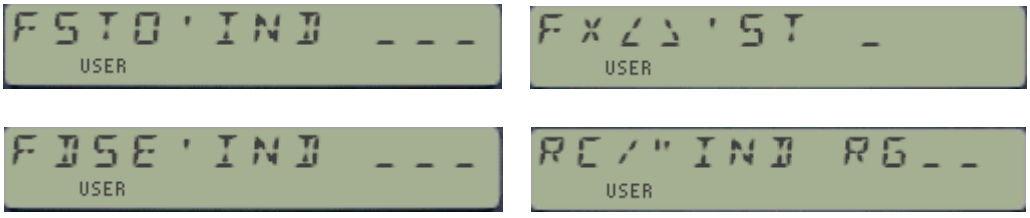
Extended Regs	Store	Recall	Other	file-Blocks
F-Register	**FSTO _ _ _**	**FRCL _ _ _**	**FX<> _ _ _**	**CLFL**
From 0 to FileSize-1	**FST+ _ _ _**	**FRC+ _ _ _**	**FVEW _ _ _**	**GETR/X**
	FST- _ _ _	**FRC- _ _ _**	**FDSE _ _ _**	**SAVER/X**
	FST* _ _ _	**FRC* _ _ _**	**FISG _ _ _**	**"FSHFT**
	FST/ _ _ _	**FRC/ _ _ _**	**POSDF**	**"RDUMP**
ALPHA	**FAST _ _ _**	**FARC _ _ _**		**_**

Besides the direct access, you also have the INDirect addressing capabilities implemented on the expanded registers; the sixteen Stack registers (including synthetic regs {M-e}); and all the standard-Registers - a hybrid mode, unique to this implementation.

Most of the functions will prompt for the parameters to use. The initial prompt is a three-field underscore for the F-register indeed. Pressing [SHIFT] changes it to IND three-digit fields for another F-register to be used as indirect. Pressing the [RADIX] key changes to the IND ST _ prompt, where you'll enter the register mnemonic, from T to e (all sixteen are available). Pressing the radix key again changes to a IND RG_ _ prompt where you can enter a standard register number to use as indirect address. Repeat pressings of the radix key act as a toggle between those two. There's also provision for direct stack and standard register arguments – even if those can be redundant in practice, being exactly the same as the original ones.

Once you complete the entry adding the register number the action is performed in RUN mode, or two program lines are entered in program mode – automatically selecting the appropriate parameter depending on the direct or indirect types. This is automatically done so you needn't (and shouldn't) edit the value entered in the program's second line at all – which will be properly interpreted in a running program.

You can move between the functions while the prompts are up; not only to select the math operation but also to change the main function amongst the group. So for instance during the **FRCL** _ _ _ main prompt pressing the [SST] key will trigger the **FX<>** function, or pressing [STO] will invoke the **FSTO** function instead. Also *you can revert to the original mainframe functions* pressing the corresponding key of the function in the prompt, for instance here pressing [RCL] will trigger the original RCL _ _



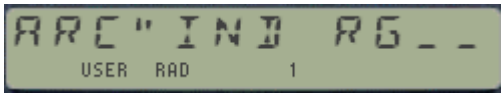
The functions will not allow you to enter values greater than the theoretical data file maximum size of 600, either as direct or indirect index. Attempting to enter larger values will trigger a "NONEXISTENT" error message. At execution time *the F-register value will be checked against the actual file size*, as F-Reg cannot be larger than FileSize-1. The same check is made for IND_RG combinations, as there's no telling at that point about how many standard registers will be available at the program editing stage.

The usage of standard stack and data registers is not only more convenient from the usability standpoint, but also it enables the RCL math on these registers via the **FRCL** function:

Although possible, it is however not meant to be used in a program because of the obviously higher byte count. That's why when used in an editing program the direct stack and data register F-functions *revert automatically to the native STO/RCL functions instead*, which has the additional benefit of a clearer representation by the OS as merged lines. *This includes the STO_Math functions as well.*

Storing and Recalling ALPHA Data

The extended functions **FAST** and **FARC** provide the means to store and recall ALPHA data directly in the FILE registers area. Like their numeric counterparts, they support direct, INDirect, stack and standard registers indexes for a complete palette of options at your disposal. You can access these directly from the FSTO/FRCL prompts by pressing the ALPHA key at any time.



*Note that these functions are somehow restricted because the ALPHA register needs to eiehter be empty or hold the data file name, so effectively **FAST** can only save that name, whilst **FARC** will always append the F-register value to it.*

Searching for a value in a Data File.

POSDF will search the Data File whose name is in ALPHA (or the current one if empty) looking for a value equal to the contents in the X-Register. X is saved in LastX. If the value is found the file record number is returned to X. Otherwise the value -1 is left in X. It is therefore analogous to **POSFL** for ASCII files.

Deleting Expanded Registers.

No additional MCODE functions are provided to clear blocks of F-registers. You can use the X-Function **CLFL** and **CLXM** from the X-Functions and AMC_OS.X modules respectively.

Other Block Operations.

No additional MCODE functions are provided to manage blocks of F-registers. You can use the X-Functions **GETR**, **GETRX**, **SAVER** and **SAVERX** from the X-functions module for these purposes.

Moving around the File Registers Functions.

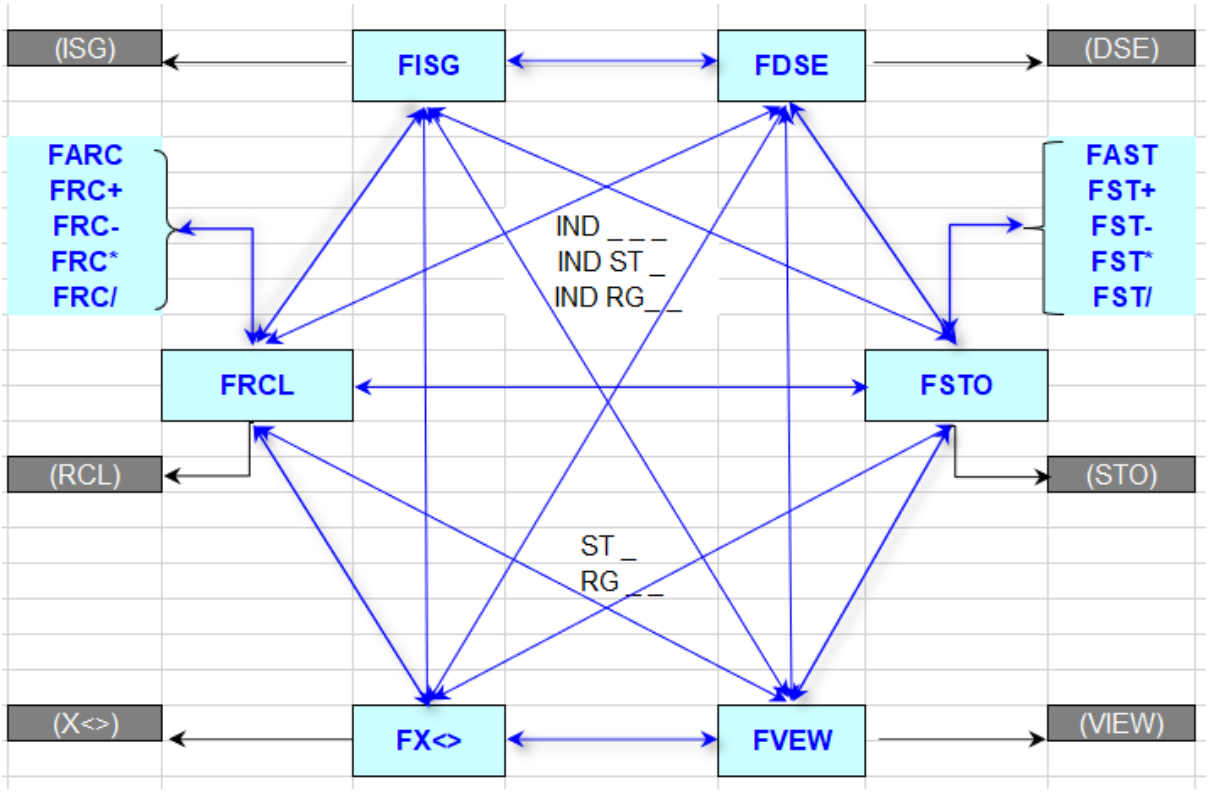


Figure 2.

Even if there isn't a dedicated launcher for these functions, navigation amongst them is as easy as intuitive. First off, assign one of the functions to its "natural" key, for instance **FSTO** to the STO key.

Then while the FSTO _ _ _ prompt is shown you can move about all the X-Reg functions by pressing the key for the corresponding action, i.e.

- RCL will toggle to FRCL _ _ _
- SST (for X<>) will launch FX<> _ _ _
- R/S (for VIEW) will launch FVEW _ _ _
- CHS (for ISG) will launch FISG _ _ _
- ALPHA will trigger FAST _ _ _
- The math keys will launch the corresponding math function, ie. FST+ _ _ _
- EEX (for DSE)will launch FDSE _ _ _
- SHIFT will add the IND _ _ prompt
- RADIX will add the ST _ prompt. All 16 status regs are selectable.
- RADIX again to toggle between ST _ and RG _ _. Choose any standard reg up to 99.
- STO again to exit to the native STO function (no way back!)
- Note that the DIRECT Stack/REG prompt is not strictly needed – that's the native function already. However, the RCL Math functions are useful and are available using this approach.

All the options above are available from within any of the 16 functions – regardless of which one you used to start the sequence. See the descriptions earlier in this manual for more details.

A few (more) Housekeeping Routines.

I've added a few more Input/output routines for convenience, including data movement between the standard and FILE registers zones. These few new routines are briefly described below:

- **WORKFL** appends the name of the current file to ALPHA. Very convenient for ease of operation of the routines, avoids having to type the file name after ALPHA changes.
- **RDUMP** copies all standard registers into the F-Regs within a Data File. It assumes that the data file is large enough to hold them, or "END OF FL" will occur.
- **FSHFT** does a selective copy of data registers to the data file whose name is in ALPHA; using the control word bbb.eee:ii in X - also assuming the data file is large enough to hold the selected number of data registers. Note that if negative then the direction of the copy is reversed, i.e. will go from the File-Area to the standard registers. In this case, make sure the SIZE is set so that there are enough standard registers to receive the File-data!
- **DFED** is a Data File editor routine to sequentially enter or visualize the values respectively. Requires the file name in ALPHA. It employs F-Regs functions itself.
- **FLRAN** populates the data file whose name is in ALPHA with random numbers, using **RNDM** from the OSX module (which takes its initial seed from the Time Module). Useful to test the sorting programs amongst other things.
- **FLSORT** will do a descending bubble sort on the data file whose name is in ALPHA. User flag 10 can be set to visualize the indexes as the program runs. Note that *this is a very slow program*; use the PPC versions **FS2** and **FS3** for speed. Numeric values only!

Below is the listing of the data movement routines. Note that **FSHFT** is more capable than SAVERX in that you can provide an increment step in the control word in X.

```
01  LBL "RDUMP"
02  FLSIZE
03  SIZE?
04  X<Y?
05  GTO 03
06  "NO ROOM"
07  AVIEW
08  WORKFL
09  STOP
10  LBL 03
11  E
12  -
13  E3
14  /
15  SAVER          ;
16  RTN
17  LBL "FSHFT"    ; bbb.eee:ii
18  CF 00
19  X<0?
20  SF 00
21  ABS
22  RUNNING
23  LBL 00
24  FC? 00
25  GTO 01
26  FRCL IND ST X(3)
27  STO IND Y(2)
28  GTO 02
29  LBL 01
30  RCL IND X(3)
31  FSTO IND ST Y(2)
32  LBL 02
33  RDN
34  ISG X(3)
35  GTO 00
36  CLD
37  END
```

Program listing for **FLRAN** and **FLSORT** is shown below.

```
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  FRCL IND RG 00
14  LBL 01
15  FS? 10
16  VIEW 01
17  FRCL IND RG 01
18  X>Y?
19  GTO 02
20  FSTO IND RG 00
21  X<>Y
22  FSTO IND RG 01
23  LBL 02
24  RDN
25  ISG 01
26  GTO 01
27  ISG 00
28  GTO 00
29  CLD
30  RTN
31  LBL "FLRAN"
32  FLSIZE
33  E
34  –
35  E3
36  /
37  LBL 03
38  RNDM
39  FSTO IND RG Y(2)
40  RDN
41  ISG X(3)
42  GTO 03
43  RDN
44  END
```

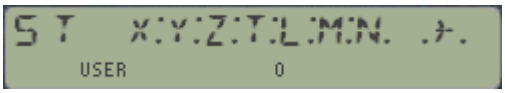
Program listing for **DFED** is shown below. Note that after each prompt the ALPHA register is reset with the file name, thus you can leave the editing at any step – no need to complete the execution till the end of file.

```
01  LBL "DFED"
02  FLSIZE
03  E
04  –
05  E3
06  /
07  LBL 00
08  FRCL IND ST X(3)
09  X<>Y
10  "D"
11  ARCLI
12  >"="
13  ARCL Y(2)
14  >"?"
15  CF 22
16  XEQ 02          ; show & reset
17  STOP
18  FC?C 22         ; entered data?
19  GTO 01          ; no, skip over
20  FSTO IND ST Y(2)
21  RDN
22  LBL 01
23  ISG X(3)
24  GTO 00
25  "DONE"
26  LBL 02
27  AVIEW
28  CLA
29  WORKFL
30  END
```

III. Stack Registers Swaps. { STKSWP }

A set of 15 functions to perform stack and data register swaps is included in this module. They are the logical extension of the native function **X<>**, but applied to all the other 15 stack registers – including the ALPHA components {M,N,O,P} and the system-reserved {a,b,c,d,e, K, Q}.

This many functions won't fit in the already-crowded FAT, therefore they have been implemented as sub-functions on an auxiliary FAT. To access them you can use any of the two sub-function launchers, either by their indexes (**TF#**) or their names (**TF$**). And in addition to this method, the header function **STKSWP** also doubles as another launcher, dedicated to this set.



;


The table below lists all sub-functions for the **STKSWP** launcher, also including the sub-index number when used as argument for **TF#**

#	Hotkey	Sub-function
1	**[A]**	**a<> _ _**
2	**[B]**	**b<> _ _**
3	**[C]**	**c<> _ _**
4	**[D]**	**d<> _ _**
5	**[E]**	**e<> _ _**
6	**[K]**	**}-<> _ _**
7	**[Y]**	**Y<> _ _**
8	**[Z]**	**Z<> _ _**

#	Hotkey	Sub-function
9	**[T]**	**T<> _ _**
10	**[L]**	**L<> _ _**
11	**[M]**	**M<> _ _**
12	**[N]**	**N<> _ _**
13	**[O]**	**O<> _ _**
14	**[P]**	**P<> _ _**
15	**[Q]**	**Q<> _ _**
-	**[X]**	Calls **X<> _ _**

Hotkey	Action
[USER]	Toggels to YMEM
[PRGM]	Calls **TF#**
[ALPHA]	Calls **TF$**
[SHIFT]	**IND** *registers*
[RADIX]	STACK registers
[EEX]	Adds "1_ _" to field
[+]	Upwards rotation
[-]	Downwards rotation

Individual Function arguments and hot-keys

Like the native **X<>**, all these functions offer full support of INDirect and STack registers as arguments. They are also prompting (despite being sub-functions), and can be used in manual (RUN) or program modes. In a program, the non-merged functions approach is extended to three program steps, which include the launcher **TF#** plus its index, and then another number for the argument.

In addition to the numeric, STACK, and IND register arguments you can also use the hot-keys [EEX], [+] and [-] to lengthen the prompt field (so you can directly address register above 99); and to rotate between all the 16 choices while their prompt is up in the LCD, regardless of which one you start off with. The sequence ends on both ends with **X<>** (one before **a<>** and one after **Q<>**), as the "end of the line", so to speak.

Remember that *you'll need to manually add 112 (70 hex) for stack register arguments;* or *128 (80 hex) for Indirect registers; or 240 (F0 hex) for both the combined case, i.e. IND_ST arguments.*

For example, to enter the function Z<> IND 05 in a program you'd use the following three steps, which is equivalent in functionality and number of bytes to the standard-functions listed on the right:

01 **TF#**	01 X<> Z
02 **8** (eighth-sub-function)	02 X<> IND 05
03 **133** (five plus 128 for indirect)	03 X<> Z

Note that these control keys are consistently used in all modules that feature auxiliary FATs for the same purposes, as follows:

- PRGM accesses the numeric launcher, T**F# _ _** (prompts for index)
- ALPHA accesses the Alphanumeric Launcher, T**F$ _"** (prompts for name)
- RADIX accesses the **LASTF** feature to recall the last-executed sub-function

Sub-function Catalog.

CAT+	**Sub-functions Catalog**	Ángel Martin	Source: Library #4
XEQ ‘	**Direct Execution**		
LASTF	**Last Function**		

The module includes an auxiliary FAT with a set of 22 sub-functions. Because these are not in the main FAT the OS knows nothing about them, so they cannot be called using XEQ, nor can they be assigned to keys using ASN. Therefore, a dedicate way to access them must exist. Two functions are available to call sub-functions either alphabetically (by spelling their name in ALPHA) or numerically, by entering their corresponding index# in the prompt:

or:

Like all other modules with sub-functions, there is a way to enumerate them using **FCAT** – itself a sub-function included in the auxiliary FAT. We already saw that **FCAT** can be triggered from the F**CAT** prompt, pressing [ENTER^], spelling its name with **TF$**, or using index #000 with **TF#**.

A few hot-keys allow you to control and navigate the catalog during the enumeration:

- [R/S] stops and resumes the listing
- [SST] manually advances to the next sub-function
- [SHIFT] reverses the direction of the show
- [XEQ] will execute the sub-function shown if the enumeration is suspended.
- Back-arrow will exit the catalog

Note that [ENTER^] would move to the next section if there were any, but that’s not the case here.

When you execute a sub-function using the launchers or by means of the **CAT+** shortcut, its index# is saved in the LastFunction buffer automatically. This allows subsequent re-execution through the LASTF facility, pressing **CAT”1** followed by the [RADIX] key.

All sub-functions are programmable. When they’re entered into a program its name is briefly shown in the display and two program steps are added by the launchers – one with the numeric launcher **TF#** and another one following it with the corresponding index#. This is done automatically for you, no need to enter it manually.

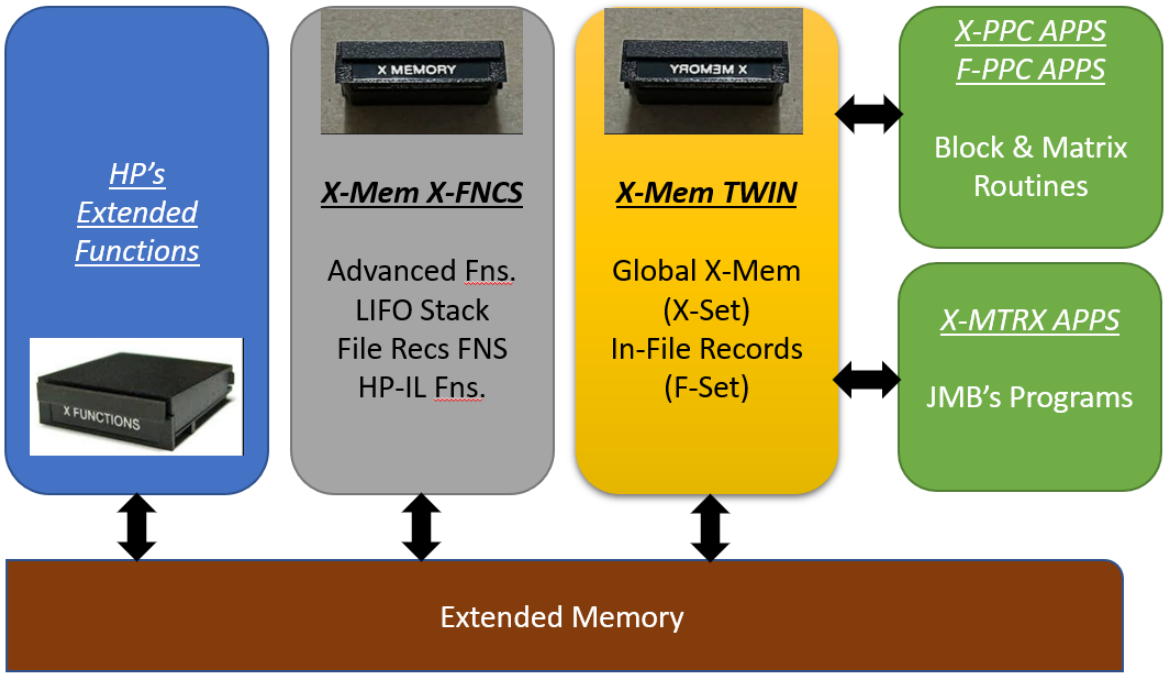
Be careful if a sub-function follows a branching test function, such as **X=Y?** – as obviously the non-merged structure of sub-functions will not be compatible with the “skip if false” rule in this circumstance. You need to work-around those cases using the negated logic and a static go-to. There are no limitations in the other way around, i.e. a test function can follow a sub-function without any issues.

CAT+ will print the sub-functions names if a printer is connected in NORM or TRACE modes.
The complete list of sub-functions is provided at the beginning of the manual.

X-Mem Modules: The Complete Picture..

The "X-Mem_TWIN" is the second extension module dedicated to management of X-Memory, above and beyond HP's original X-Functions/Memory Module (later included in the HP-41 CX). You're encouraged to also check the "X-Mem X-Functions" module that includes many advanced functions and new uses for X-Memory.

The figure below shows the complete picture of modules and their interdependencies:



Comparative Function Summary: Data management

Functionaliry	Mainframe / X-Fns	X-Mem X-FNCS	XM TWIN
Data File Read	GETX, GETR, GETRX	FLRCL, FLVEW	FRCL, FRC- , FRC+, FRC*, FRC/, FVEW
Data File Write	SAVEX, SAVER, SAVERX	FLSTO, FLX<>	FSTO, FST-, FST+, FST*, FST/, FX<>
Data Find	POSA, POSFL		POSDF, XFINDX
Data Regs Management	CLX, CLRG, CLRGX, REGMOVE, REGSWAP	CLXM, CLMM	CLEM, XCLREG, CLRGX XRGMOV, XRGSWP
Data Regs Recall	RCL, ARCL	ARCLIP	XRCL, XRC-, XRC+, XRC*, XRC/, XARC
Data Regs Storage	STO, ST-, ST+, ST*, ST/, X<>, ASTO		XSTO, XST-, XST-, XST*, XST/, XAST
Sort, View & Exchange	VIEW, X<>Y, X<>	SORTFL, A<>RG, A<>ST, ST<>RG	XVEW, XX<>, A<>XRG, ST<>XRG

A few MCODE listings.

The following MCODE listings are for the simpler register manipulation functions **CLXRG**, **A<>XRG** and **ST<>XRG**.

The first one deals with the three X-Mem blocks independently, so the main routine calls the register clearing routine three times, providing the from/to parameters in A.X and B.X

1	CLXRG	*Header*	A486	*087*	*"G"*	
2	CLXRG	*Header*	A487	*012*	*"R"*	***Clear Extended Registers***
3	CLXRG	*Header*	A488	*018*	*"X"*	*ALL of them!*
4	CLXRG	*Header*	A489	*00C*	*"L"*	
5	CLXRG	*Header*	A48A	*003*	*"C"*	*Ángel Martin*
6	CLXRG	**CLXRG**	**A48B**	130	LDI S&X	
7	CLXRG		A48C	***301***	***bottom of XM-2***	
8	CLXRG		A48D	0E6	C<>B S&X	
9	CLXRG		A48E	130	LDI S&X	
10	CLXRG		A48F	***3F0***	***Top of X-MEM2***	
11	CLXRG		A490	106	A=C S&X	
			A491	*379*	*PORT DEP:*	*Clear XM Block*
1,007	EM-2/0	3EF	A492	*03C*	*XQ*	*from A.X to B.X*
			A493	*0A3*	*->A4A3*	*[CLRBLK]*
769		301	A494	130	LDI S&X	
			A495	***201***	***bottom of XM-1***	
751	EM-1/0	2EF	A496	0E6	C<>B S&X	
			A497	130	LDI S&X	
513		201	A498	***2F0***	***Top X-MEM1***	
511	MAIN-0	1FF	A499	106	A=C S&X	
			A49A	*379*	*PORT DEP:*	*Clear XM Block*
			A49B	*03C*	*XQ*	*from A.X to B.X*
			A49C	*0A3*	*->A4A3*	*[CLRBLK]*
192		0C0	A49D	130	LDI S&X	
191	XF/M-0	0BF	A49E	***040***	***bottom of XM-0***	
64		040	A49F	0E6	C<>B S&X	
			A4A0	130	LDI S&X	
16	Status/0	00F	A4A1	***0BF***	***Top X-MEM0***	
			A4A2	106	A=C S&X	
30	CLXRG	**CLRBLK**	**A4A3**	**1A6**	**A=A-1 S&X**	*starts at "3FF"*
31	CLXRG		A4A4	0A6	A<>C S&X	
32	CLXRG		A4A5	270	RAMSLCT	*select register*
33	CLXRG		A4A6	106	A=C S&X	*keep rg adr in A.X*
34	CLXRG		A4A7	04E	C=0 ALL	
35	CLXRG		A4A8	2F0	WRTDATA	*clear it*
36	CLXRG		A4A9	326	?A<B S&X	*last one?*
37	CLXRG		A4AA	*3CB*	*JNC -07*	*no, do next*
38	CLXRG		A4AB	270	RAMSLCT	*select chip0*
39	CLXRG		A4AC	3E0	RTN	

The next listing shows how to deal with the exchange of a block of five X-Registers and either the Stack or ALPHA. There we take advantage of the fact that both ALPHA and the Stack have fixed addresses, which facilitates the task.

Also listed is the code for **CLXRGX,** more complicated as it needs to deal with the infamous X-Mem gaps. This has been managed by using HP's original code for CLRGX adding the two subroutines [CHKBND] and [NOVOID], for bound-checking and void-avoidance respectively. Slippery indeed, but it'll get even more so with **XRGMIV** and **XRGSWP** the worst of all.

1	AST<>YRG	*Header*	A9E8	*087*	*"G"*	
2	AST<>YRG	*Header*	A9E9	*012*	*"R"*	
3	AST<>YRG	*Header*	A9EA	*018*	*"X"*	***Swap Alpha & Y-Regs***
4	AST<>YRG	*Header*	A9EB	*03E*	*">"*	*PROMPTING*
5	AST<>YRG	*Header*	A9EC	*13C*	*"<"*	
6	AST<>YRG	*Header*	A9ED	*101*	*"A"*	*Ken Emery*
7	AST<>YRG	**A<>XRG _ _ _**	**A9EE**	**088**	**SETF 5**	
8	AST<>YRG		A9EF	*04B*	*JNC +09*	
9	AST<>YRG	*Header*	A9F0	*087*	*"G"*	
10	AST<>YRG	*Header*	A9F1	*012*	*"R"*	
11	AST<>YRG	*Header*	A9F2	*018*	*"X"*	***Stack swap Y-REG***
12	AST<>YRG	*Header*	A9F3	*03E*	*">"*	*PROMPTING*
13	AST<>YRG	*Header*	A9F4	*03C*	*"<"*	
14	AST<>YRG	*Header*	A9F5	*114*	*"T"*	
15	AST<>YRG	*Header*	A9F6	*113*	*"S"*	*Ángel Martin*
16	AST<>YRG	**ST<>XRG _ _ _**	**A9F7**	**084**	**CLRF 5**	
17	AST<>YRG	MERGE	A9F8	04C	?FSET 4	
18	AST<>YRG		A9F9	*01F*	*JC +03*	*SST'ing a program*
19	AST<>YRG		A9FA	2CC	?FSET 13	
20	AST<>YRG		A9FB	*01B*	*JNC +03*	*RUN'ing a program*
21	AST<>YRG		A9FC	*179*	*?NC XQ*	*Get Parameter from NextLine*
22	AST<>YRG		A9FD	*10C*	*->435E*	*[GETRG#]*
23	AST<>YRG		A9FE	130	LDI S&X	
24	AST<>YRG		A9FF	***004***	***4 registers***	*block size*
25	AST<>YRG		AA00	146	A=A+C S&X	*{L,X,Y,Z,T}*
26	AST<>YRG		AA01	*3B5*	*PORT DEP:*	*Check Bounds*
27			AA02	*08C*	*XQ*	*and adjust address*
28	**T <-> Rnn+4 <-> M**		AA03	*0F2*	*->ACF2*	*[CHKBND]*
29	**Z <-> Rnn+3 <-> N**		AA04	39C	PT= 0	*To use as a counter*
30	**Y <-> Rnn+2 <-> O**		AA05	046	C=0 S&X	*Starts in register 0(T)*
31	**X <-> Rnn+1 <-> P**		AA06	***08C***	***?FSET 5***	
32	**L <-> Rnn <-> Q**		AA07	*01B*	*JNC +03*	
33			AA08	130	LDI S&X	
34	AST<>YRG		AA09	005	CON: 5	*Starts in register 5(M)*
35	AST<>YRG		AA0A	0E6	C<>B S&X	*Stack address in B.X*
36	AST<>YRG	LOOP5	AA0B	0A6	A<>C S&X	*Source block address in C.X*
37	AST<>YRG		AA0C	106	A=C S&X	*keep it in A.X*
38	AST<>YRG		AA0D	130	LDI S&X	
39	AST<>YRG		AA0E	***1FF***	***CON:***	
40	AST<>YRG		AA0F	366	?A#C S&X	is addr = 0x0C0 ?
41	AST<>YRG		AA10	***027***	***JC +04***	no, ignore
42	AST<>YRG		AA11	130	LDI S&X	*yes, need to skip the "void"*
43	AST<>YRG		AA12	140	***CON:***	*void size*
44	AST<>YRG		AA13	1C6	A=A-C S&X	bump it to 0x0BF
45	AST<>YRG		AA14	0A6	A<>C S&X	
46	AST<>YRG		AA15	270	**RAMSLCT**	*Select register Block*
			AA16	106	A=C S&X	*keep rg. Block addr in A.X*
1,007	EM-2/0	3EF	AA17	038	**READATA**	*Read Source Register*
			AA18	070	**N=C ALL**	*store register content in N*
769		301	AA19	0C6	C=B S&X	*recall stack rg addr*
			AA1A	270	RAMSLCT	*Select stack reg*
751	EM-1/0	2EF	AA1B	0E6	C<>B S&X	*save stack reg adr n A*
			AA1C	038	**READATA**	*read stack rg content*
513		201	AA1D	0F0	C<>N ALL	*exchange stack content w/ RG con*
511		1FF	AA1E	2F0	**WRTDATA**	*Write RG content in stack rg*
			AA1F	0C6	C=B S&X	
	MAIN-0		AA20	226	C=C+1 S&X	*increase stack rg. Pointer*
			AA21	0E6	C<>B S&X	*Keep it in B.X*
192		0C0	AA22	0A6	A<>C S&X	*recall RG block add to C*
191	XF/M-0	0BF	AA23	270	RAMSLCT	*select RG block again*
64		040	AA24	106	A=C S&X	*save RG block addr in A.X*
			AA25	0F0	C<>N ALL	
16	Status/0	00F	AA26	2F0	**WRTDATA**	*write stack content to RG block*
			AA27	1A6	A=A-1 S&X	*decrease RG block addr*
66	AST<>YRG		AA28	3DC	PT=PT+1	*increase counter*
67	AST<>YRG		AA29	094	?PT= 5	*Carry if finished*
68	AST<>YRG		AA2A	*30B*	*JNC -31d*	*[LOOP5]*
69	AST<>YRG		AA2B	3E0	RTN	

1	CLXRGX	*Header*	A9A5	*098*	*"X"*	
2	CLXRGX	*Header*	A9A6	*007*	*"G"*	***Clrear Register Range***
3	CLXRGX	*Header*	A9A7	*012*	*"R"*	***bbb,eee|ii***
4	CLXRGX	*Header*	A9A8	*018*	*"X"*	*limited to XR <= 604*
5	CLXRGX	*Header*	A9A9	*00C*	*"L"*	
6	CLXRGX	*Header*	A9AA	*003*	*"C"*	*Ángel Martin*
7	CLXRGX	**CLXRGX**	**A9AB**	**0F8**	**READ 3(X)**	
8	CLXRGX		A9AC	0EE	B<>C ALL	save in B
9	CLXRGX		A9AD	0CE	C=B ALL	keep it in C
10	CLXRGX		A9AE	*171*	*?NC XQ*	*check for integer/ALPHA*
11	CLXRGX		A9AF	*0C8*	*->325C*	*[LB_325C]*
12	CLXRGX		A9B0	070	N=C ALL	
13	CLXRGX		A9B1	**204**	**CLRF 2**	
14	CLXRGX		A9B2	*339*	*?NC XQ*	*GET 1ST. 3 FRAC DIGIT OF X*
15			A9B3	*0D0*	*->34CE*	*[GTFRAB]*
16	IF CF5, THEN: N[2:0]=EEE		A9B4	106	A=C S&X	
17	N[5:3]= REG ADDR OF BBB		A9B5	0B0	C=N ALL	
18	N[8:6]= REG ADR OF RRR		A9B6	1BC	RCR 11	
19			A9B7	0A6	A<>C S&X	
20	CLXRGX		A9B8	070	N=C ALL	
21	CLXRGX		A9B9	**208**	**SETF 2**	*WILL DECODE X AS RRR.BBB---*
22	CLXRGX		A9BA	*339*	*?NC XQ*	*N[5:3]= RRR, N[2:0]= BBB*
23	CLXRGX		A9BB	*0D0*	*->34CE*	*[GTFRAB]*
24	CLXRGX		A9BC	2E6	?C#0 S&X	
25	CLXRGX		A9BD	*017*	*JC +02*	*LB_5904*
26	CLXRGX		A9BE	226	C=C+1 S&X	
27	CLXRGX		A9BF	106	A=C S&X	
28	CLXRGX		A9C0	0B0	C=N ALL	
29	CLXRGX		A9C1	1BC	RCR 11	
30	CLXRGX		A9C2	0A6	A<>C S&X	
31	CLXRGX		A9C3	03C	RCR 3	
32	CLXRGX		A9C4	070	N=C ALL	
33	CLXRGX		A9C5	106	A=C S&X	
34	CLXRGX		A9C6	*3B5*	*PORT DEP:*	*Check A.X for Bounds*
35	CLXRGX		A9C7	*08C*	*XQ*	*and adjust address*
36	CLXRGX		A9C8	*0F2*	*->ACF2*	*[CHKBND]*
37	CLXRGX		A9C9	0B0	C=N ALL	
38	CLXRGX		A9CA	0A6	A<>C S&X	
39	CLXRGX		A9CB	03C	RCR 3	
40	CLXRGX		A9CC	070	N=C ALL	
41	CLXRGX		A9CD	106	A=C S&X	
42	CLXRGX		A9CE	*3B5*	*PORT DEP:*	*Check A.X for Bounds*
43	CLXRGX		A9CF	*08C*	*XQ*	*and adjust address*
44	CLXRGX		A9D0	*0F2*	*->ACF2*	*[CHKBND]*
45	CLXRGX		A9D1	0B0	C=N ALL	
46	CLXRGX		A9D2	0A6	A<>C S&X	
47	CLXRGX		A9D3	1BC	RCR 11	
48	CLXRGX		A9D4	226	C=C+1 S&X	
49	CLXRGX		A9D5	070	N=C ALL	
50	CLXRGX		A9D6	03C	RCR 3	
51	CLXRGX		A9D7	106	A=C S&X	
52	CLXRGX		A9D8	13C	RCR 8	
53	CLXRGX		A9D9	0E6	B<>C S&X	*increment to B.X*
54	CLXRGX	NXTREG	A9DA	*36D*	*PORT DEP:*	*check addr isn't in main RAM*
55	CLXRGX		A9DB	*08C*	*XQ*	*offseet by 320 if so*
56	CLXRGX		A9DC	*0C1*	*->A4C1*	*[NOVOID]*
57	CLXRGX		A9DD	04E	C=0 ALL	
58	CLXRGX		A9DE	0A6	A<>C S&X	
58			A9DF	270	RAMSLCT	
58	*this can be problematic,*		A9E0	0A6	A<>C S&X	
58	*A+B could go into the main RAM*		A9E1	**2F0**	**WRTDATA**	*clear register*
58	*and there's no easy way to trap it*		A9E2	0B0	C=N ALL	
58	*as the increment is not "1".*		A9E3	126	A=A+B S&X	add increment
58			A9E4	306	?A<C S&X	reached/passed the limit?
58	CLXRGX		A9E5	*3AF*	*JC -11d*	*n0, do next -> LB_591C*
58	CLXRGX		A9E6	3E0	RTN	yes, we're done.

with the subroutines as follows:

[CHKBND] first checks that the X-Reg index is less that 605, bailing out with an error if not. Then it calculates the memory address for the register, skipping the memory gaps appropriately to ensure we won't be writing into voids or (even worse) the main RAM area.

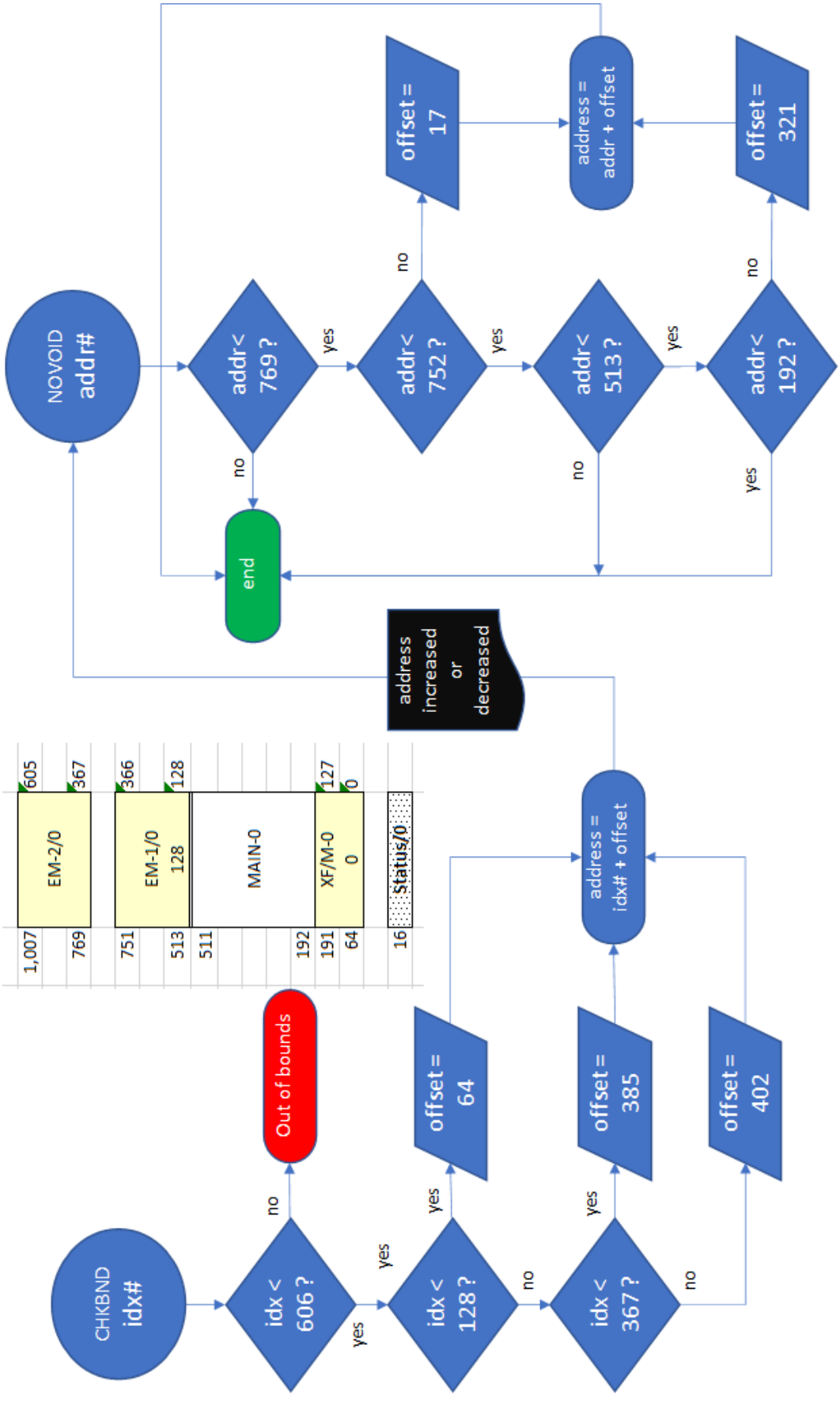
373	XSTORE	CHKBND	ACF2	130	LDI S&X	
374			ACF3	25E	maximum value = 605	2AF is the maximum
375	from index to address		ACF4	306	?A<C S&X	ilegal index?
376			ACF5	381	?NC GO	yes, abort function
377	XSTORE		ACF6	00A	->02E0	[ERRNE]
378	XSTORE		ACF7	130	LDI S&X	
379	XSTORE		ACF8	080	lower top = 128	
380	XSTORE		ACF9	306	?A<C S&X	is index < 128 ?
381	XSTORE		ACFA	023	JNC +04	N0, -> [OFFST2]
382	XSTORE	OFFST1	ACFB	130	LDI S&X	
383	XSTORE		ACFC	040	offset to addr = 64	(64)
384	XSTORE		ACFD	053	JNC +10d	[ABSADR]
385	XSTORE	OFFST2	ACFE	130	LDI S&X	
386	XSTORE		ACFF	16F	lower top = 367	
387	XSTORE		AD00	306	?A<C S&X	is index < 367
388	XSTORE		AD01	023	JNC +04	N0, indxex >= 367 -> [OFFST3]
389	XSTORE	OFFST3	AD02	130	LDI S&X	yes, add 385 to index for address
390	XSTORE		AD03	181	offset to addr = 385 d	(64 + 321)
391	XSTORE		AD04	01B	JNC +03	
392	XSTORE	ABSADR	AD05	130	LDI S&X	add 402 to index for address
393	XSTORE		AD06	192	offset3 to addr = 402 d	(64+321+17)
394	XSTORE	ADJUST	AD07	146	A=A+C S&X	

The [NOVOID] is also needed to check that the address used is not inside of a void or main RAM. This is needed in addition to checking the bounds, because some routines modify the target addresses using increments on the parameters supplied; so even if these are veted we must double check those derived from them.

1	NOVOID	NOVOID	A4C1	130	LDI S&X	
2	NOVOID		A4C2	301	CON: 769	
3	NOVOID		A4C3	306	?A<C S&X	is addr < 301 ?
4	NOVOID		A4C4	3A0	?NC RTN	NO, addr >= 301 then do nothing
5	NOVOID		A4C5	130	LDI S&X	
6	NOVOID		A4C6	2F0	CON: 752	
7	NOVOID		A4C7	306	?A<C S&X	is addr < 2F0 ?
8	NOVOID		A4C8	027	JC +04	if addr < 2F0 then KEEP CHECKING
9	NOVOID		A4C9	130	LDI S&X	NO, it needs bumping
10	NOVOID		A4CA	011	CON: 17	offse by (17)
11	NOVOID		A4CB	05B	JNC +11d	add it to address
12	NOVOID		A4D6	130	LDI S&X	
13	NOVOID		A4CD	201	CON: 513	
14	NOVOID		A4CE	306	?A<C S&X	is addr < 201 ?
15	NOVOID		A4CF	3A0	?NC RTN	NO, addr >= 201 then do nothing
16	NOVOID		A4D0	130	LDI S&X	
17	NOVOID		A4D1	0C0	CON: 192	
18	NOVOID		A4D2	306	?A<C S&X	is addr < 0C0 ?
19	NOVOID		A4D3	360	?C RTN	YES, ido nothing
20	NOVOID		A4D4	130	LDI S&X	no. it needs bumping
21	NOVOID		A4D5	141	CON: 321	skip main RAM
22	NOVOID		A4D6	146	A=A+C S&X	
23	NOVOID		A4D7	3E0	RTN	

The good news is that with these two subroutines alone we'll be able to complete the remaining, more complex functions **XRGMOV** and **XRGSWP**, with no need for yet more additions to HP's base code.

See below the flowcharts showing graphically the adjustments made by these subroutines.



Final Bonus: Copying code from bank-switched ROMS.

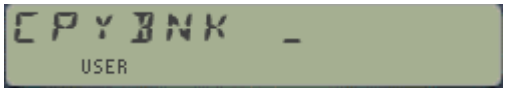
Here's a last-minute addition to the module – not related to the Extended memory subject but rather interesting per-se.

There are almost no tools available to extract or copy code from a bank-switched ROM. When faced with that challenge I typically used ad-hoc modifications of Warren Furlow's routine **CB**, posted at: http://www.hp41.org/LibView.cfm?Command=View&ItemID=317

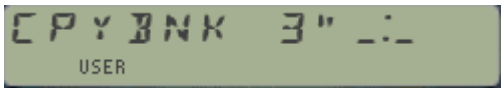
That routine is specific for fixed source and destination pages, as well as only useful for the second bank. Writing a more general-purpose function was always on my mind, and finally here it is at last.

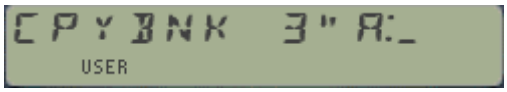
CPYBNK is a prompting function. It has a customized prompt with three distinct sections that are shown on the screen as the data entry progresses. The parameters entered are as follows:

- Bank number, an integer decimal from 1 to 4
- Source page, an hex value from 0 to F
- Destination page, same as above.



The function is smart enough to know the valid range of each prompted value, thus it'll simply ignore non-allowed values, presenting the same prompt again. You can use the back-arrow key to cancel at any moment. Once the bank number is entered the prompt requests the "FROM:TO" pages, as denoted by the underscore characters on both sides of the colon. The screens below show this at different stages of the process:





The copy is always made into the main bank of the destination page (bank-1). This is typically a Q-RAM page in an MLDL (or a RAM page on the CL) thus only supports one bank. Besides the practical usage is intended to copy elusive, hard-to-reach code buried into secondary banks – therefore it wouldn't appear very sensible to copy it into equally obscure destinations.

The main bank is the first one; therefore you can use "1" to select it. In this case the function does the same as **CPYPGE** in the PowerCL, or **COPYROM** in the HEPAX.

If the source ROM doesn't have the chosen bank an error message is shown and the execution aborts. More than just a convenient feature, this is vital to ensure that the execution doesn't activate a non-existing bank – which could create all kinds of havoc if the location of the missing bank is already occupied in RAM or FLASH by other modules.

There is no restriction made to the choice of pages. The function will read whatever is in the source (or zeroes if nothing) and will attempt to write it on the destination. Obviously to be successful the destination must be a Q-RAM (MLDL or CL).

CPYBNK Source Code.

Here it is for your enjoyment, not a complex piece of code but tricky just the same. The only specific details to keep in mind are the fixed locations within each page reserved for the bank-switch instructions; as well as the convention followed in the page signature characters (the trailing text). Both are used by the routine to make sure it's ok to execute the switching command.

1	CPYBNK	*Header*	A878	*08B*	*"K"*	
2	CPYBNK	*Header*	A879	*00E*	*"N"*	
3	CPYBNK	*Header*	A87A	*002*	*"B"*	***bank# in prompt***
4	CPYBNK	*Header*	A87B	*019*	*"Y"*	***From:TO in prompt***
5	CPYBNK	*Header*	A87C	*310*	*"P"*	
6	CPYBNK	*Header*	A87D	*103*	*"C"*	*Ángel Martin*
7	CPYBNK	**CPYBNK**	**A87E**	**000**	**NOP**	
8	CPYBNK		A87F	346	?A#0 S&X	
9	CPYBNK		A880	*02B*	*JNC +05*	
10	CPYBNK		A881	130	LDI S&X	*ONLY 4 BANKS MAX*
11	CPYBNK		A882	005	CON:	*so 5 is the limit*
12	CPYBNK		A883	306	?A<C S&X	n<5?
13	CPYBNK		A884	***03F***	***JC +07***	yes, go on
14	CPYBNK		A885	130	LDI S&X	*re-exec the function!*
15	CPYBNK		A886	***31B***	***function code***	*A5:1B*
16	CPYBNK		A887	236	C=C+1 XS	*"41B"*
17	CPYBNK		A888	236	C=C+1 XS	*"51B"*
18	CPYBNK		A889	*329*	***?NC GO***	*Check #0 and [RAK70]*
19	CPYBNK		A88A	*132*	*->4CCA*	*[RAK704]*
20	CPYBNK		A88B	0A6	A<>C S&X	*get prompt input to C[S&X]*
21	CPYBNK		A88C	***128***	***WRIT 4(L)***	*store bank# in LastX*
22	CPYBNK		A88D	*3D1*	*?NC XQ*	*Right Justify LCD - Enables LCD*
23	CPYBNK		A88E	*118*	*->46F4*	*[RIGHTJ]*
24	CPYBNK		A88F	130	LDI S&X	
25	CPYBNK		A890	***022***	***" " "***	*Double Quotes*
26	CPYBNK		A891	***3E8***	***WRIT 15(e)***	*write it in display (9-bit)*
27	CPYBNK		A892	130	LDI S&X	
28	CPYBNK		A893	***09F***	***" _:"***	*Underscore w/colon*
29	CPYBNK		A894	***3E8***	***WRIT 15(e)***	*as new prompt*
30	CPYBNK	NOSHFT1	A894	*329*	*?NC XQ*	*Inputs Hex key - 0-F, SHIFT*
31	CPYBNK		A894	*120*	*->48CA*	*[HEXKEY] - from B1 only*
32	CPYBNK		A894	2C6	?B#0 S&X	*was it SHIFT?*
33	CPYBNK		A898	***3EB***	***JNC -03***	*yes, ignore and repeat*
34	CPYBNK		A899	*3B8*	*READ 14(d)*	*remove excess prompt*
35	CPYBNK		A89A	***0C6***	***C=B S&X***	*copy chr$ to C[S&X]*
36	CPYBNK		A89B	3D8	C<>ST XP	
37	CPYBNK		A89C	288	SETF 7	add colon
38	CPYBNK		A89D	3D8	C<>ST XP	
39	CPYBNK		A89E	***3E8***	***WRIT 15(e)***	*write it in display (9-bit)*
40	CPYBNK		A89F	*379*	*PORT DEP:*	***Get pg# from Key in B{S&X}***
41	CPYBNK		A8A0	*03C*	*XQ*	*returns pg# in A[S&X]*
42	CPYBNK		A8A1	*0FE*	*->A8FE*	*[KEYPG]*
43	CPYBNK		A8A2	*149*	*?NC XQ*	*valid return*
44	CPYBNK		A8A3	*024*	*->0952*	*[ENCP00]*
45	CPYBNK		A8A4	04E	C=0 ALL	
46	CPYBNK		A8A5	0A6	A<>C S&X	
47	CPYBNK		A8A6	13C	RCR 8	move it to C<6>
48	CPYBNK		A8A7	**268**	**WRIT 9(Q)**	*source page, pg#*

```
49  CPYBNK                  A8A8  3D9  ?NC XQ          Enable but not clear LCD
50  CPYBNK                  A8A9  01C  ->07F6          [ENLCD]
51  CPYBNK  NOSHFT2         A8AA  329  ?NC XQ          Inputs  Hex key - 0-F, SHIFT
52  CPYBNK                  A8AB  120  ->48CA          [HEXKEY] - from B1 only
53  CPYBNK                  A8AC  2C6  ?B#0 S&X         was it SHIFT?
54  CPYBNK                  A8AD  3EB  JNC -03          yes, ignore and repeat
55  CPYBNK                  A8AE  0C6  C=B S&X          copy chr$ to C[S&X]
56  CPYBNK                  A8AF  3E8  WRIT 15(e)       write it in display (9-bit)
57  CPYBNK                  A8B0  379  PORT DEP:        Get pg# from Key in B{S&X}
58  CPYBNK                  A8B1  03C  XQ               returns pg# in A[S&X]
59  CPYBNK                  A8B2  0FE  ->A8FE           [KEYPG]
60  CPYBNK                  A8B3  0A6  A<>C S&X
61  CPYBNK                  A8B4  13C  RCR 8            move it to C<6>
62  CPYBNK                  A8B5  070  N=C ALL          destination pg#
63  CPYBNK                  A8B6  1D5  ?NC XQ          LeftJ, Test, EnRAM & Reset SEQ
64  CPYBNK                  A8B7  118  ->4675          [CLNUP4]
65  CPYBNK  CHKBSM          A8B8  006  A=0 S&X          reset field
66  CPYBNK                  A8B9  1A6  A=A-1 S&X        "FFF"
67  CPYBNK                  A8BA  138  READ 4(L)        bank#
68                          A8BB  1C6  A=A-C S&X        subtract it from 'FFF"
69  No need to check for bk1 A8BC  266  C=C-1 S&X       bk#-1
70                          A8BD  266  C=C-1 S&X        bk#-2
71  CPYBNK                  A8BE  0C7  JC  +24d         no need to switch
72  CPYBNK                  A8BF  278  READ 9(Q)        get source pg# to C[S&X]
73                          A8C0  03C  RCR 3            move it to C<3>
74  first we check that the bank
                            A8C1  0A6  A<>C S&X
75  is marked in the ROM signature
                            A8C2  1BC  RCR 11           put in [ADR] field
76  (as a pre-requisite)    A8C3  330  FETCH S&X        read marker
77                          A8C4  2F6  ?C#0 XS          bank marked?
78  CPYBNK                  A8C5  14B  JNC +41d         no, abort
79  CPYBNK  BSWITCH         A8C6  138  READ 4(L)        bank#
80  CPYBNK                  A8C7  266  C=C-1 S&X        bk#-3
81                          A8C8  02B  JNC +05
82  then we send the execution
                            A8C9  379  PORT DEP:        Check forCode 2 & Switch
83  to the proper switching point,
                            A8CA  03C  XQ               in the source module
84  also checking the code! A8CB  107  ->A907           [CHKCD2]
85                          A8CC  053  JNC +10          [DATA]
86  CPYBNK                  A8CD  266  C=C-1 S&X        bk#-4
87  CPYBNK                  A8CE  02B  JNC +05
88  CPYBNK                  A8CF  379  PORT DEP:        Check forCode3 & Switch
89  CPYBNK                  A8D0  03C  XQ               in the source module
90  CPYBNK                  A8D1  110  ->A910           [CHKCD3]
91  CPYBNK                  A8D2  023  JNC +04          [DATA]
92  CPYBNK                  A8D3  379  PORT DEP:        Check forCode4 & Switch
93  CPYBNK                  A8D4  03C  XQ               in the source module
94  CPYBNK                  A8D5  119  ->A919           [CHKCD4]
95  CPYBNK  DATA            A8D6  0B0  C=N ALL
96  CPYBNK                  A8D7  0EE  C<>B ALL         destination pg# in B<6>
97  CPYBNK                  A8D8  278  READ 9(Q)        get source pg# to C[S&X]
98  CPYBNK                  A8D9  03C  RCR 3            move it to C<3>
99  CPYBNK  LOOP            A8DA  1BC  RCR 11           move it to C<6>
100 CPYBNK                  A8DB  330  FETCH S&X        read word
101 CPYBNK                  A8DC  15C  PT= 6
102 CPYBNK                  A8DD  0E2  C<>B @PT         destination page
103 CPYBNK                  A8DE  040  WROM             write it in destination
104 CPYBNK                  A8DF  0E2  C<>B @PT         restore source pg#
```

105	CPYBNK		A8E0	03C	RCR 3	move to S&X field
106	CPYBNK		A8E1	226	C=C+1 S&X	next word
107	CPYBNK		A8E2	***3C3***	***JNC -08***	loop back
108	CPYBNK		A8E3	***138***	***READ 4(L)***	*bank#*
109	CPYBNK		A8E4	266	C=C-1 S&X	*bk#-1*
110	CPYBNK		A8E5	266	C=C-1 S&X	*was bk#=1?*
111	CPYBNK		A8E6	*3C1*	*?C GO*	*yes, do a proper exit*
112	CPYBNK		A8E7	*003*	*->2BF7*	*[NFRPU]*
113	CPYBNK	BKSWCH1	A8E8	278	READ 9(Q)	*no, get SOURCE pg#*
114	CPYBNK		A8E9	09C	PT= 5	
115	CPYBNK		A8EA	3D0	LD@PT- F	*location for BK1:*
116	CPYBNK		A8EB	310	LD@PT- C	*"pFC7"*
117	CPYBNK		A8EC	1D0	LD@PT- 7	
118	CPYBNK		A8ED	***1E0***	***GOTO ADR***	***switch back to bank-1***
119	CPYBNK	NOBANK	A8EE	***138***	***READ 4(L)***	*bank#*
120	CPYBNK		A8EF	0EE	C<>B ALL	
121	CPYBNK		A8F0	*321*	*?NC XQ*	*Show "NO_" msg*
122	CPYBNK		A8F1	*10C*	*->43C8*	*[NOMSG4]*
123	CPYBNK		A8F2	*002*	***"B"***	
124	CPYBNK		A8F3	*001*	***"A"***	*"NO BANK"*
125	CPYBNK		A8F4	*00E*	***"N"***	
126	CPYBNK		A8F5	*00B*	***"K"***	
127	CPYBNK		A8F6	*220*	***" "***	
128	CPYBNK		A8F7	06E	A<>B ALL	
129	CPYBNK		A8F8	01E	A=0 MS	
130	CPYBNK		A8F9	17E	A=A+1 MS	
131	CPYBNK		A8FA	*3A1*	*?NC XQ*	*Generate dec. number ->display!*
132	CPYBNK		A8FB	*014*	*->05E8*	*[GENNUM]*
133	CPYBNK		A8FC	*1F1*	*?NC GO*	*LeftJ, Show and Halt*
134	CPYBNK		A8FD	*0FE*	*->3F7C*	*[APEREX]*
135	CPYBNK	**KEYPG**	**A8FE**	**066**	**A<>B S&X**	*put page# in A[S&X]*
136	CPYBNK		A8FF	31C	PT= 1	*clean up parameter:*
137	CPYBNK		A900	342	?A#0 @PT	*from chr# to page#*
138	CPYBNK		A901	*027*	*JC +04*	
139	CPYBNK		A902	130	LDI S&X	*A[S&X] goes from 1 to 6*
140	CPYBNK		A903	009	CON:	*need to add 9 to chr#*
141	CPYBNK		A904	146	A=A+C S&X	*it now ranges from A to F*
142	CPYBNK		A905	002	A=0 @PT	*clear the "3" digit!*
143	CPYBNK		A906	3E0	RTN	
144	CPYBNK	**CHKCD2**	**A907**	130	LDI S&X	
145	CPYBNK		A908	180	CON:	*ENBNK2 code*
146	CPYBNK		A909	106	A=C S&X	save in A for compares
147	CPYBNK		A90A	278	READ 9(Q)	*get SOURCE Pg#*
148	CPYBNK		A90B	09C	PT= 5	
149	CPYBNK		A90C	3D0	LD@PT- F	*location for BK2:*
150	CPYBNK		A90D	310	LD@PT- C	*"pFC9"*
151	CPYBNK		A90E	250	LD@PT- 9	
152	CPYBNK		A90F	***093***	***JNC +18d***	*[READCD]*
153	CPYBNK	**CHKCD3**	**A910**	130	LDI S&X	
154	CPYBNK		A911	140	CON:	*ENBNK3 code*
155	CPYBNK		A912	106	A=C S&X	save in A for compares
156	CPYBNK		A913	278	READ 9(Q)	*get SOURCE Pg#*
157	CPYBNK		A914	09C	PT= 5	
158	CPYBNK		A915	3D0	LD@PT- F	*location for BK3:*
159	CPYBNK		A916	310	LD@PT- C	*"pFC3"*
160	CPYBNK		A917	0D0	LD@PT- 3	

161	CPYBNK		A918	***04B***	***JNC +09***	*[READCD]*
162	CPYBNK	**CHKCD4**	**A919**	130	LDI S&X	
163	CPYBNK		A91A	1C0	CON:	*ENBNK4 code*
164	CPYBNK		A91B	106	A=C S&X	save in A for compares
165	CPYBNK		A91C	278	READ 9(Q)	*get SOURCE Pg#*
166	CPYBNK		A91D	09C	PT= 5	
167	CPYBNK		A91E	3D0	LD@PT- F	*location for BK4:*
168	CPYBNK		A91F	310	LD@PT- C	*"pFC5"*
169	CPYBNK		A920	150	LD@PT- 5	
170	CPYBNK	READCD	A921	**330**	**FETCH S&X**	*read word*
171	CPYBNK		A922	366	?A#C S&X	*does it match code?*
172	CPYBNK		A923	***25F***	***JC -53d***	no, abort!
173	CPYBNK		A924	***1E0***	***GOTO ADR***	***switch bank***

Off topic: Ulam's Conjecture

Completely off-topic subject but it sort of happened while preparing this manual – what an excuse, uh?
Reference: https://en.wikipedia.org/wiki/Collatz_conjecture

The **ULAM** sub-function does a complete path starting with the value in X, all the way until the end when "1" is reached using the well-known Ulam's (or Collatz's) algorithm:

If odd, multiply by three and add one
If even, divide by two

$$f(n) = \begin{cases} \frac{n}{2} & \text{if } n \equiv 0 \pmod 2 \\ 3n+1 & \text{if } n \equiv 1 \pmod 2. \end{cases}$$

The function will take the integer part of the absolute value of the number in X. Then all intermediate values are briefly shown, and the total number of "nodes" is left in X upon completion. The starting number is left in X.

Examples:

41, XEQ "T$" "ULAM" -> generates a sequence of 109 numbers
22, ULAM -> generates a sequence of 15 numbers

The sequence for n = 27, listed below, takes 111 steps (41 steps through odd numbers), climbing as high as 9232 before descending to 1.

27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
(sequence A008884 in the OEIS)

ULAM Source Code

Header	A8EF	*0CD*	*"M"*	
Header	A8F0	*041*	*"A"*	
Header	A8F1	*04C*	*"L"*	
Header	A8F2	*055*	*"U"*	
ULAM	**A8F3**	**0F8**	**READ 3(X)**	
	A8F4	128	WRIT 4(L)	
	A8F5	*149*	*?NC XQ*	*Integer & Positive*
	A8F6	*134*	*->4D52*	*[CHKZI]*
	A8F7	268	WRIT 9(Q)	
	A8F8	04E	C=0 ALL	
	A8F9	0E8	WRIT 3(X)	*reset the counter*
LOOP1	A8FA	00E	A=0 ALL	
	A8FB	35C	PT= 12	*Builds "1" in A*
	A8FC	162	A=A+1 @PT	
	A8FD	278	READ 9(Q)	
	A8FE	36E	?A#C ALL	*end of the path?*
	A8FF	**3A0**	**?NC RTN**	*yes, end here.*
	A900	0F8	READ 3(X)	
	A901	2A0	SETDEC	
	A902	*01D*	*?NC XQ*	*increase counter*
	A903	*060*	*->1807*	*[AD2_10]*
	A904	0E8	WRIT 3(X)	*update value*
	A905	278	READ 9(Q)	*get current n*
	A906	*3CD*	*?NC XQ*	*C= MOD[int(C),2]*
	A907	*100*	*->40F3*	*[MOD2]*
	A908	2EE	?C#0 ALL	*it is odd?*
	A909	***02F***	***JC +05***	*yes, skip*
	A90A	278	READ 9(Q)	
EVEN	A90B	*3CD*	*?NC XQ*	*{A,B} = {C} /2*
	A90C	*13C*	*->4FF3*	*[DIVTWO]*
	A90D	***053***	***JNC +10d***	*show result*
ODD	A90E	04E	C=0 ALL	
	A90F	35C	PT= 12	
	A910	0D0	LD@PT- 3	
	A911	10E	A=C ALL	
	A912	278	READ 9(Q)	
	A913	*135*	*?NC XQ*	*3*n*
	A914	*060*	*->184D*	*[MP2_10]*
	A915	*001*	*?NC XQ*	*3*n+1*
	A916	*060*	*->1800*	*[ADDONE]*
MERGE	A917	268	WRIT 9(Q)	
	A918	*099*	*?NC XQ*	*Sends C to display - sets HEX*
	A919	*02C*	*->0B26*	*[DSPCRG]*
	A91A	*1FD*	*?NC XQ*	*wait a little - CL compatible*
	A91B	*12C*	*->4B7F*	*[WAIT4L] - Enables RAM*
	A91C	*1FD*	*?NC XQ*	*wait a little - CL compatible*
	A91D	*12C*	*->4B7F*	*[WAIT4L] - Enables RAM*
	A91E	*2E3*	*JNC -36d*	*[LOOP1]*

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, Key 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. Also it's unfortunately not possible to change their locations to other pages in RAM, say 1kB higher (for a second set of XM).

@ 0x401:

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

@ 0x301:

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

X-Mem TWIN Application ROMS

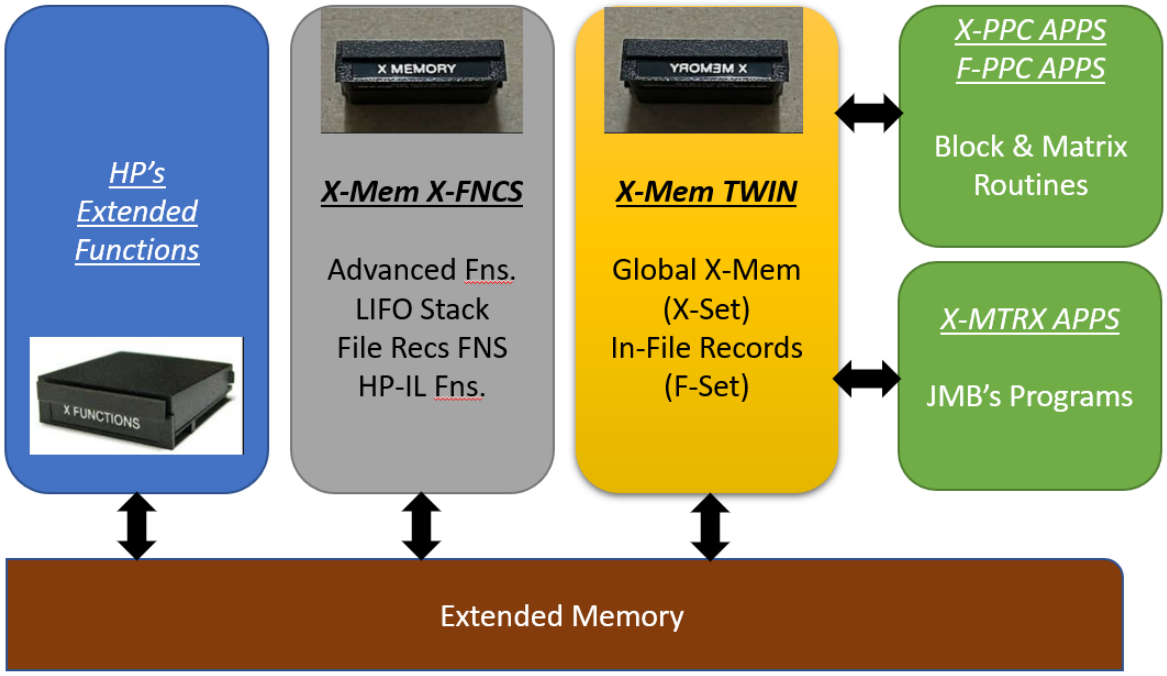
There are two application ROMS based on the X-Mem_TWIN module:

- The X-PPC APPS includes all Block and Matrix routines from the PPC ROM, plus a few application programs published in the PPC ROM Manuals. There are independent sets of routines using the global X-Mem registers and the In-File Records. You should refer to the PPC User's Manual for user instructions and description of the routines included in the X-PPC APPS ROM.

- The X-MTRX APPS includes a comprehensive set of Matrix programs from Jean-Marc Baillard collection, see:
  http://hp41programs.yolasite.com/matrixop.php,
  http://hp41programs.yolasite.com/eigen.php, and
  http://hp41programs.yolasite.com/determinant.php

The X-Mem_TWIN is the second extension module dedicated to management of X-Memory, above and beyond HP's original X-Functions/Memory Module (later included in the HP-41 CX). You're encouraged to also check the X-Mem X-Functions module that includes many advanced functions and new uses for X-Memory.

The figure below shows the complete picture of modules and their interdependencies:



X- PPC Routines. (adapted from PPC ROM).

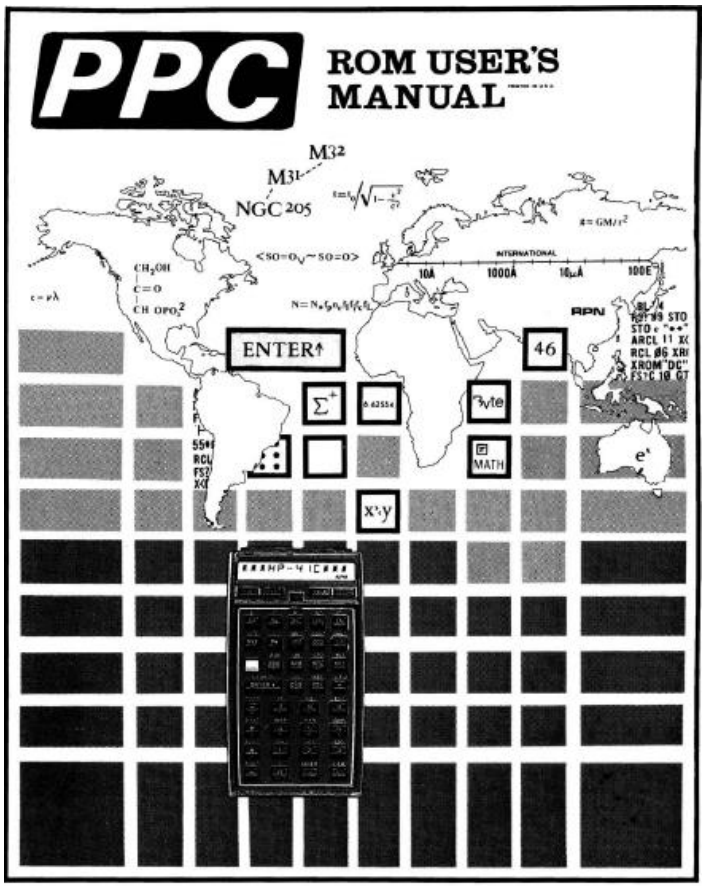
Below are the routines included in the X-PPC APPS ROM - a collection based on the block & Matrix programs from the PPC ROM adapted to the X-Mem framework, and thus using the functions form the XMem-TWIN Module.

#	Function	Description	Dependency	Type	Author
0	**-XPPC ALPHA**	*Section Header*	n/a	MCODE	*n/a*
1	**"ACMP**	Alphabetizer	XMTWIN	FOCAL	
2	**"AORD**	Alpha Order	XMTWIN	FOCAL	
3	**"XAL**	Alphabetize X & Y	XMTWIN	FOCAL	
4	**"XAM**	Alpha to X-Mem	XMTWIN	FOCAL	
5	**"XMA**	X-Mem to Alpha	XMTWIN	FOCAL	
6	**"NC**	N-th. Character	XMTWIN	FOCAL	
7	**"SU**	Substitute Character	XMTWIN	FOCAL	
8	**-XPPC MTRX**	*Section Header*	*n/a*	MCODE	*n/a*
9	**"QR**	Quotient Remainder	XMTWIN	FOCAL	
10	**"XMIO**	X-Matrix Input/Output	XMTWIN	FOCAL	
11	**"XRRM**	Row Reduction X-Matrix	XMTWIN	FOCAL	
12	**"XM1**	Interchange Two Rows	XMTWIN	FOCAL	
13	**"XM2**	Multiply row by constant	XMTWIN	FOCAL	
14	**"XM3**	Add multiple of row to another	XMTWIN	FOCAL	
15	**"XM4**	X-Register addr to (I, j)	XMTWIN	FOCAL	
16	**"XM5**	(I, j) to X-Register address	XMTWIN	FOCAL	
17	**"XS1**	Stack Sort	XMTWIN	FOCAL	
18	**"XS2**	Small X-Size Sort (n<33)	XMTWIN	FOCAL	
19	**"XS3**	Large X-Size Sort (n>32)	XMTWIN	FOCAL	
20	**-XPPC BLOCK**	*Section Header*	*n/a*	MCODE	*n/a*
21	**"XBC**	X-Block Clear	XMTWIN	FOCAL	
22	**"XBE**	X-Block Exchange	XMTWIN	FOCAL	
23	**"XBI**	X-Block Increment	XMTWIN	FOCAL	
24	**"XBM**	X-Block Move	XMTWIN	FOCAL	
25	**"XBR**	X-Block Rotate	XMTWIN	FOCAL	
26	**"XBS**	X-Block Statistics	XMTWIN	FOCAL	
27	**"XBV**	X-Block View	XMTWIN	FOCAL	
28	**"XBX**	X-Block Extrema	XMTWIN	FOCAL	
29	**"XDR**	Delete X-Record	XMTWIN	FOCAL	
30	**"XIR**	Insert X-Record	XMTWIN	FOCAL	
31	**"XMS**	**X-Memory to Stack**	XMTWIN	FOCAL	
32	**"XPR**	Pack X-Registers	XMTWIN	FOCAL	
33	**"XSM**	Stack to X-Memory	XMTWIN	FOCAL	
34	**"XUR**	Unpack X-Registers	XMTWIN	FOCAL	
35	**-FPPC MTRX**	*Section Header*	*n/a*	MCODE	*n/a*
36	**"FMIO**	F-Matrix Input/Output	XMTWIN	FOCAL	
37	**"FRRM**	Row Reduction F-Matrix	XMTWIN	FOCAL	
38	**"FM1**	Interchange Two Rows	XMTWIN	FOCAL	
39	**"FM2**	Multiply row by constant	XMTWIN	FOCAL	
40	**"FM3**	Add multiple of row to another	XMTWIN	FOCAL	
41	**"FM4**	F-Register addr to (I, j)	XMTWIN	FOCAL	
42	**"FM5**	(I, j) to F-Register address	XMTWIN	FOCAL	
43	**"FS1**	Stack Sort	XMTWIN	FOCAL	
44	**"FS2**	Small F-Size Sort (n<33)	XMTWIN	FOCAL	
45	**"FS3**	Large F-Size Sort (n>32)	XMTWIN	FOCAL	
46	**-FPPC BLOCK**	*Section header*	*n/a*	MCODE	*n/a*
47	**"FBC**	X-Block Clear	XMTWIN	FOCAL	
48	**"FBE**	F-Block Exchange	XMTWIN	FOCAL	

49	**"FBI**	F-Block Increment	XMTWIN	FOCAL
50	**"FBM**	F-Block Move	XMTWIN	FOCAL
51	**"FBR**	F-Block Rotate	XMTWIN	FOCAL
52	**"FBS**	F-Block Statistics	XMTWIN	FOCAL
53	**"FBV**	F-Block View	XMTWIN	FOCAL
54	**"FBX**	F-Block Extrema	XMTWIN	FOCAL
55	**"FDR**	Delete F-Record	XMTWIN	FOCAL
56	**"FIR**	Insert F-Record	XMTWIN	FOCAL
57	"FMS	F-Records to Stack	XMTWIN	FOCAL
58	**"PFR**	Pack F-Records	XMTWIN	FOCAL
59	**"FSM**	Stack to F-Records	XMTWIN	FOCAL
60	**"FUR**	Unpack F-Registers	XMTWIN	FOCAL
61	**"FAM**	F-Records to Alpha	XMTWIN	FOCAL
62	**"FMA**	Alpha to F-Records	XMTWIN	FOCAL
63				

As you can see the module has parallel sections on global X-Memory Registers and In-File Records. The function naming for functions on those sets is only different in the first character: X for global and F for "ïn-file" – which should make it easier to remember and identify.

You're encouraged to refer to the PPC User's Manual for descriptions and user instructions for the routines.



X- Matrix Routines. (adapted from Jean-Marc Baillard programs).

Below are the routines included in the XMATRIX APPS ROM - a collection based on Jean-Marc Baillard's Matrix programs adapted to the X-Mem framework, and thus using the functions form the XMen-TWIN Module.

#	Function	Description	Dependency	Type	Author
0	**-JMB XMATRX**	*Lib#4 Check & Splash*	Lib#4	MCODE	*Nelson F. Crowle*
1	**?XMSQ**	Tests if matrix is Square	Lib#4	MCODE	*Ángel Martin*
2	**M*M**	X-Matrix Product	Lib#4	MCODE	*JM Baillard*
3	**MNORM**	X-Matrix Norm	Lib#4	MCODE	*JM Baillard*
4	**MTRACE**	X-Matrix Trace	Lib#4	MCODE	*JM Baillard*
5	**"CRXMAT**	Create X-Matrix	XMTWIN	FOCAL	*JM Baillard*
6	**"XDET**	X-Matrix Determinant	XMTWIN	FOCAL	*JM Baillard*
7	**"XDFL**	Eigenvalues by Deflation	XMTWIN	FOCAL	*JM Baillard*
8	**"XLS3 _**	Linear Systems	XMTWIN	FOCAL	*JM Baillard*
9	**"XM-**	Element Subtraction	XMTWIN	FOCAL	*JM Baillard*
10	**"XM+**	Element Addition	XMTWIN	FOCAL	*JM Baillard*
11	**"XM***	Element Product	XMTWIN	FOCAL	*JM Baillard*
12	**"XM/**	Element Division	XMTWIN	FOCAL	*JM Baillard*
13	**"XM=**	Copies X-Matrix	XMTWIN	FOCAL	*Ángel Martin*
14	**"XM^2**	X-Matrix Squared	XMTWIN	FOCAL	*Ángel Martin*
15	**XMCO**	Matrix Copy	XMTWIN	FOCAL	*JM Baillard*
16	**"XMINV**	X-Matrix Inversion	XMTWIN	FOCAL	*JM Baillard*
17	**"XMEXP**	X-Matrix Exponential	XMTWIN	FOCAL	*JM Baillard*
18	**"XMLN**	X-Matrix Logarithm	XMTWIN	FOCAL	*JM Baillard*
19	**"XMCHP**	X-Mat. Characteristic Polyn	XMTWIN	FOCAL	*JM Baillard*
20	**"XMP**	X-Matrix Product	XMTWIN	FOCAL	*JM Baillard*
21	**"XMPL**	X-Matrix Polynomial	XMTWIN	FOCAL	*JM Baillard*
22	**"XMPOW**	X-Metrix n-th. Power	XMTWIN	FOCAL	*JM Baillard*
23	**"XMRAN**	Random X-Matrix	XMTWIN	FOCAL	*JM Baillard*
24	**"XMRCL**	X-Matrix Recall	XMTWIN	FOCAL	*Ángel Martin*
25	**"XMSTO**	X-Matrix Store	XMTWIN	FOCAL	*Ángel Martin*
26	**"XMROOT**	X-matrix p-th. Root	XMTWIN	FOCAL	*JM Baillard*
27	**"XMSQRT**	X-Matrix Square Root	XMTWIN	FOCAL	*JM Baillard*
28	**"XMTRN**	X-Matrix Transpose	XMTWIN	FOCAL	*JM Baillard*
29	**"XMZRO**	Clears All Elements	XMTWIN	FOCAL	*Ángel Martin*
30	**"XPMIN**	Minimum Polynomial	XMTWIN	FOCAL	*JM Baillard*
31	**"XRANM**	**Makes a Random Mtrx**	XMTWIN	FOCAL	*JM Baillard*
32	**"XRNSYM**	**Symmetric Random Mtrx**	XMTWIN	FOCAL	*JM Baillard*
33	**"XONE**	Makes all-ones X-Matrix	XMTWIN	FOCAL	*Ángel Martin*
34	**"XIDN**	Makes Identity X-Matrix	XMTWIN	FOCAL	*Ángel Martin*
35	**"XZDG**	Zeroes X-Matrx Diagonal	XMTWIN	FOCAL	*Ángel Martin*

What follows is the description and usage instructions for the programs.
You can also refer to Jean-Marc's website for more details; see:

http://hp41programs.yolasite.com/matrixop.php,
http://hp41programs.yolasite.com/eigen.php, and
http://hp41programs.yolasite.com/determinant.php

Storing and recalling an X-Matrix.

The next programs are for the storage and review of the matrix. The input required is the control word, and the element enumeration will proceed in column order as mentioned before.

These routines use the standard registers {R00 - R03} as auxiliary for control – contrary to the Advantage-style matrices there's no header containing index information, thus that needs to be done using standard registers.

As you can see the *control word* is returned to X upon completion of the data input/review. This is your handle to the matrix, thus the importance to have it available for the subsequent operation. It is also stored in R02 in case you need it.

1	LBL "YMSTO"			36	STO 00	*1.0rr*
2	SF 01			37	LBL 00	
3	***GTO 01***			38	RCL 00	*k.00rr*
4	LBL "YMRCL"			39	LBL 02	
5	CF 01			40	*"a"*	
6	LBL 01			41	ARCLI	*"ak"*
7	STO 02	*bbb.eee.rr*		42	*"/-,"*	*"ak,"*
8	FRC	*0.eeerr*		43	RCL 01	*p,0cc*
9	***E3***			44	ARCLI	*"ak,p"*
10	*	*eee.rr*		45	*"/-="*	*"ak,p="*
11	INT	*eee*		46	RDN	
12	RCL 02	*bbb.eee.rr*		47	YARC	*IND RG_06*
13	INT			48	2051	
14	STO 03	*bbb*		49	FS? 01	
15	-	*eee-bbb*		50	*"/-?"*	*"ak,p= xxxxxxx?"*
16	***E***			51	AVIEW	
17	+	*eee-bbb+1*		52	FC? 01	
18	RCL 02	*bbb.eee.rr*		*53*	***GTO 01***	
19	***E3***			54	CF 22	
20	*	*bbbeee.rr*		55	STOP	
21	FRC	*0.rr*		56	FC?C 22	
22	***E2***			57	GTO 01	
23	*			58	YSTO	*IND RG_06*
24	STO 00	*rr*		59	2051	
25	/	*cc = (eee-bbb+1)/rr*		60	LBL 01	
26	***E3***			61	FC? 01	
27	/	*0.0cc*		62	PSE	
28	***E***			63	ISG 03	*next Y-register*
29	+			64	NOP	
30	STO 01	*1.0cc*		65	ISG X	*next row*
31	RCL 00	*rr*		*66*	***GTO 02***	
32	***E3***			67	ISG 01	*next column*
33	/			*68*	***GTO 00***	
34	***E***			69	RCL 02	*control word*
35	+			70	END	*all done.*

Matrix Element Arithmetic

The four routines below operate on the individual matrix elements, using them to populate a new result matrix where each element is the result of the arithmetic operation on the elements of the source matrices.

```
XM-   ->   cij = aij - bij    ;    XM+   ->   cij = aij + bij
XM*   ->   cij = aij * bij    ;    XM/   ->   cij = aij / bij
```

If the source matrices don't have the same dimension only the "common" sub-matrices will be used for the calculation of the result matrix.

The routines require the control words of the source matrices in Z and Y, plus the first resister of the destination matrix in the X-register.

STACK	INPUTS	OUTPUT
Z	bbb.eeerr1	/
Y	bbb.eeerr2	/
X	bbb3	bbb.eeerr3

Example: A = 3 1 4 / 1 5 9 / R01 R03 R05 = R02 R04 R06 and B = 2 7 1 / 8 2 8 / R07 R09 R11 = R08 R10 R12 respectively

The control number of A is 1.00602
The control number of B is 7.01202 if, for instance, you choose to store A+B in registers R15 .....

1.00602 ENTER^, 7.01202 ENTER^, 15 XEQ "M-" =>> 15.02002 and

A-B = 1 -6 3 / -7 3 1 = R15 R17 R19 / R16 R18 R20 respectively

Note that the destination block of registers may be the same as the one of matrix A or B – but be careful do not use a straddle block between them that can alter the source values as the results are being calculated and stored.

```
01 *LBL "XM*"
02  CF 01
03  GTO 01
04 *LBL "XM/"
05  SF 01
06 *LBL 01
07  SF 00
08  GTO 01
09 *LBL "XM+"
10  CF 02
11  GTO 00
12 *LBL "XM-"
13  SF 02
14 *LBL 00
15  CF 00
16 *LBL 01
17  STO M
18  STO N
19  CLX
20   E3
21  *
22  INT
23   E3
24  ST/ Y
25 *LBL 02
26  CLX
27  XRCL IND Z (3073)
28  XRCL IND Y (3074)
29  FC? 00
30  GTO 01
31  FS? 00
32  1/X
33  *
34  GTO 02
35 *LBL 01
36  FS? 02
37  CHS
38  +
39 *LBL 02
40  XSTO IND M  (3077)
41  CLX
42  SIGN
43  ST+ M
44  ST+ Z
45  ISG Y
46  GTO 02
47  RCL M
48  X<>Y
49   -
50  R^
51   E3
52  *
53  FRC
54  +
55   E3
56  /
57  RCL N
58  +
59  CLA
60  END
```

Transpose of a Matrix

The transpose of a nxm matrix A = [aij] is the mxn matrix tA = [bi j] defined by bij = aji
"XTRN" stores the transpose of a matrix in a different block of registers.
The 2 blocks cannot overlap.

STACK	INPUTS	OUTPUTS
Y	bbb.eeerr1	rr1+bbb.eeerr1
X	bbb2	bbb.eeerr2

Example:

```
        2 7 1 3          R01  R04  R07  R10
A =     1 9 4 2    =     R02  R05  R08  R11
        4 6 2 1          R03  R06  R09  R12
```

and you want to store tA in registers R21 ...etc...

1.01203 ENTER^, 21 XEQ "XMTRN" =>> 21.03204 and

```
         2  1  4          R21  R25  R29
tA =     7  9  6    =     R22  R26  R30
         1  4  2          R23  R27  R31
         3  2  1          R24  R28  R32
```

XMTRN Program listing.-

```
01 *LBL "XMTRN"
02  STO O
03  RCL Y
04  STO T
05  FRC
06  ISG X
07  INT
08  STO M
09  STO N
10  RDN
11  LBL 01
12  XRCL IND Y  (3074)
13  XSTO IND Y  (3074)
14  CLX
15  SIGN
16  +
17  ISG Y
18  GTO 01
19  RDN
20  SIGN
21  +
22  ENTER^
23  R^
24  DSE N
25  GTO 01
26  STO Y
27  RCL O
28  -
29  RCL M
30  /
31  DSE Y
32  E2
33  /
34  +
35  E3
36  /
37  RCL O
38  +
39  CLA
40  END
```

Random Matrices and Register Sorting.

"XMRAN" stores pseudo-random integers (between 1 and N) into registers Rbb , ... , Ree
No other register is used.

STACK	INPUT	OUTPUT
Y	bbb.eeerr	/
X	N	/

where bbb.eeerr is the control number of the array

Example: Store random integers between 1 and 12 into registers R01 , R02 , ... , R07

1.007 ENTER^, 12 XEQ "RANM" gave:
R01 = 10 R02 = 5 R03 = 8 R04 = 6 R05 = 6 R06 = 8 R07 = 10 ...

Since the current date & time are used to initialize the random number generator, you will probably get different values.

```
01 LBL "XMRAN"
02 DATE
03 TIME
04 +
05 LBL 01
06 R-D
07 FRC
08 RCL X
09 X<> Z
10 ST* Z
11 X<> Z
12 INT
13 ISG X
14 CLX
15 XSTO IND T (3072)
16 RDN
17 ISG Z
18 GTO 01
19 END
```

Note that also included in the module is "**XRAN**". This one follows a different approach, using the RNG function **RNDM** from the AMC_OS/X module. The input parameter is an X-registers control word bbb.eee, and therefore it's a very useful way to test the sorting routine **XSORT**

Program listing.-

```
01 *LBL "XSORT"
02 RUNNING
03 SIGN
04 *LBL 01
05 LASTX
06 LASTX
07 XRCL IND L (3076)
08 *LBL 02
09 XRCL IND Y (3074)
10 X>Y?
11 GTO 03
12 X<>Y
13 LASTX
14 +
15 *LBL 03
16 RDN
17 ISG Y
18 GTO 02
19 XX<> IND L (3076)
20 XSTO IND Z (3073)
21 ISG L
22 GTO 01
23 CLD
24 RTN
25 *LBL "XRAN"
26 RCL X
27 *LBL 00
28 RNDM
29 XSTO IND Y (3074)
30 RDN
31 ISG X
32 GTO 00
33 RDN
34 END
```

Inverse of a Matrix

"MINV" can invert up to a 24 x 24 matrix using the Gauss-Jordan elimination - also called the "exchange method".

Here, the first element of the matrix must be stored into XR06.
Standard Data registers R01 thru R05 are used for control numbers of different rows and columns.

You put the order of the matrix into X-register and XEQ "MINV".
The determinant is left in X-register and in R00 and the inverse matrix has replaced the original one (in registers XR06 ... etc.)

If flag F01 is clear, Gaussian elimination with partial pivoting is used.
If flag F01 is set, the pivots are the successive elements of the diagonal.

STACK	INPUT	OUTPUT
X	n	det A

where n is the order of the matrix A

Example: Let's take the 5x5 Pascal's matrix – 6.03005, XEQ "XMSTO"

```
 1  1  1  1  1          R06  R11  R16  R21  R26
 1  2  3  4  5          R07  R12  R17  R22  R27
 1  3  6 10 15    =>    R08  R13  R18  R23  R28      respectively
 1  4 10 20 35          R09  R14  R19  R24  R29
 1  5 15 35 70          R10  R15  R20  R25  R30
```

5 XEQ "MINV" the determinant (1 in this example) will be in X-register and in R00 and the inverse matrix:

```
  5 -10  10  -5  1          R06  R11  R16  R21  R26
-10  30 -35  19 -4          R07  R12  R17  R22  R27
 10 -35  46 -27  6    =>    R08  R13  R18  R23  R28      respectively
 -5  19 -27  17 -4          R09  R14  R19  R24  R29
  1  -4   6  -4  1          R10  R15  R20  R25  R30
```

Program Listing.-

```
01 *LBL "XMINV"
02  RUNNING
03  STO 01
04  0,1
05  %
06  ST+ 01
07  STO 02
08  X<>Y
09  ST* Y
10   E-5
11  STO T
12  *
13  X<>Y
14  6.005
15  ST+ 02
16  +
17  STO 05
18  +
19  STO 03
20  +
21  STO 04
22  SIGN
23  STO 00
24  CLA
25 *LBL 14
26  FS? 01
27  GTO 01
28  CLST
29  RCL 04
30  INT
31  RCL 02
32  FRC
33  +
34  RDN
35  +
36 *LBL 00
37  CLX
38  XRCL IND Z (3073)
39  ABS
40  X<=Y?
41  GTO 00
42  X<>Y
43  ENTER^
44  +
45 *LBL 00
46  ISG Z
47  GTO 00
48  R^
```

```
49  INT
50  RCL 04
51  INT
52  -
53  RCL 03
54  STO Z
55  +
56  XTOA
57  X=Y?
58  GTO 01
59 *LBL 09
60  XRCL IND X  (3075)
61  XX<> IND Z  (3073)
62  XSTO IND Y  (3074)
63  ISG Y
64  RDN
65  ISG Y
66  GTO 09
67  RCL 00
68  CHS
60  STO 00
70 *LBL 01
71  RCL 02
72  INT
73  RCL 05
74  INT
75  X=Y?
76  GTO 03
77  RCL 03
78  INT
79  X=Y?
80  GTO 02
81  XRCL IND X  (3075)
82  XRC* IND T  (3072)
83  XRC/ IND R4 (2052)
84  XST-  IND Z  (3073)
85 *LBL 02
86  ISG 02
87  GTO 04
88  RCL 01
89  INT
90  ST- 02
91  ST+ 03
92  GTO 04
93 *LBL 03
94  RCL 01
95  INT
96  ST+ 03
97  ST+ 05
98  DSE 05
99 *LBL 04
100  ISG 05
101  GTO 01
102 *LBL 05
103  RCL 02
104  INT
105  RCL 04
106  INT
107  X=Y?
108  GTO 06
109  XRCL IND X  (3075)
110  XST/ IND Z  (3073)
111 *LBL 06
112  ISG 02
113  GTO 05
114  RCL 01
115  INT
116  X^2
117  ST- 03
118 *LBL 07
119  RCL 03
120  INT
121  RCL 04
122  INT
123  X=Y?
124  GTO 08
125  XRCL IND X (3075)
126  CHS
127  XST/ IND Z  (3073)
128 *LBL 08
129  ISG 03
130  GTO 07
131  XRCL IND R4 (2052)
132  ST* 00
133  1/X
134  XSTO IND R4  (2052)
135  RCL 01
136  FRC
137  ST+ 02
138  LASTX
139  INT
140  X^2
141  ST- 03
142  ST- 05
143  SIGN
144  ST+ 03
145  ISG 04
146  GTO 14
147  FS? 01
148  GTO 11
149  RCL 01
150  INT
151  STO 04
152 STO 05
153 *LBL 10
154  RCL 05
155   E
156  -
157  AROT
158  ATOX
159  6
160  -
161  RCL 04
162  ST* Z
163  *
164  6
165  ST+ Z
166  +
167  RCL 01
168  FRC
169  +
170   E3
171  /
172  +
173  XRGSWP
174  DSE 05
175  GTO 10
176 *LBL 11
177  RCL 00
178  CLD
179  END
```

Euclidean Norm of a Matrix

MNORM computes || A || = (Σi,j ai,j^2)/2

STACK	INPUT	OUTPUT
X	bbb.eeerr	|| A ||

Example:

```
      2 7 1 3       R01 R04 R07 R10
A =   1 9 4 2   =>  R02 R05 R08 R11
      4 6 2 1       R03 R06 R09 R12
```

1.01203 XEQ "NORM" =>> || A || = 14.8996643

Trace of a Square Matrix

The trace of a square matrix equals the sum of its diagonal elements.

STACK	INPUT	OUTPUT
X	bbb.eeerr	Tr(A)

Example:

```
      1 2 4     R03 R06 R09
A =   3 5 7  =  R04 R07 R10
      7 9 8     R05 R08 R11
```

3.01103 XEQ "TRACE" =>> Tr(A) = 14

Both MNORM and MTRACE are written in MCODE; here you can see an equivalent FOCAL routines:

```
01 LBL "MNORM"
02 E3
03 *
04 INT
05 E3
06 /
07 0
08 LBL 01
09 XRCL IND Y
10 X^2
11 +
12 ISG Y
13 GTO 01
14 SQRT
15 END
```

```
01 LBL "TRACE"
02 E-5
03 +
04 0
05 LBL 01
06 XRCL IND Y
07 +
08 ISG Y
09 GTO 01
10 END
```

Multiplication of two Matrices

M*M calculates the product of 2 matrices A & B and returns the control number o
f A*B/ The number of columns of the first matrix must equal the number of rows of the second matrix:

STACK	INPUT	OUTPUT
Z	bbb.eeerr1	rr1 = rr3
Y	bbb.eeerr2	rr1 = rr3
X	bbb3	bbb.eeerr3

Example: Calculate C = A.B where

$$A = \begin{matrix} 2 & 7 & 1 & 3 \\ 1 & 9 & 4 & 2 \\ 4 & 6 & 2 & 1 \end{matrix} \qquad B = \begin{matrix} 3 & 1 \\ 4 & 2 \\ 7 & 5 \\ 2 & 6 \end{matrix}$$

assuming that: A is stored in registers R01 thru R12
B is stored in registers R15 thru R22
and choosing R26 as the first register of C

In other words,

```
R01 R04 R07 R10       2  7  1  3         R15  R19        3  1
R02 R05 R08 R11   =   1  9  4  2   and   R16  R20   =    4  2
R03 R06 R09 R12       4  6  2  1         R17  R21        7  5
                                         R18  R22        2  6
```

Key in: 1.01203 ENTER^, 15.02204 ENTER^, 26 XEQ "M*M" =>> 26.03103

the control number of the matrix C and the result is:

$$C = \begin{matrix} 47 & 39 \\ 71 & 51 \\ 52 & 32 \end{matrix} = \begin{matrix} R26 & R29 \\ R27 & R30 \\ R28 & R31 \end{matrix}$$

M*M is also written in MCODE, here you can see an equivalent FOCAL routine:

```
01 LBL "M*M"
02 STO O
03 STO P
04 RDN
05 STO N
06 X<>Y
07 STO M
08 FRC
09 ISG X
10 INT
11 STO Q
12 LBL 01
13 CLX
14 STO IND P
15 RCL M
16 RCL N
17 LBL 02
18 XRCL IND Y
19 XRCL IND Y
20 *
21 XST+ IND P
22 CLX
23 SIGN
24 +
25 ISG Y
26 GTO 02
27 LASTX
28 ST+ M
29 ST+ P
30 DSE Q
31 GTO 01
32 RCL M
33 FRC
34 ISG X
35 INT
36 STO Q
37 ST- M
38 ISG N
39 GTO 01
40 ENTER^
41 SIGN
42 %
43 RCL P
44 LASTX
45 -
46 +
47 E3
48 /
49 RCL O
50 +
51 CLA
52 END
```

Copying a Matrix and Matrix Square Power.

"**XM=**" does a copy of the matrix with control word in the X-register. It uses the registers following immediately after the source one, without any gaps. The routine leaves the initial control word in X.

For example:

1.02505 , XEQ"XM=" => makes a copy into registers { R36 – R50 }

This is a nice and simple example because most of what the routine does is prepare the argument for the final **XRGMOV** instruction.

Matrix Square power.

"**XM^2**" is a simple application of **M*M** and uses the Matrix Copy routine above to make a copy of itself. This imposes a maximum dimension of 14x14 for the matrix to square. The routine expects the matrix to be square, so there's an initial check using function **XMSQ?** To verify this assumption.

Program listing.-

```
1     LBL "XM="     ; sss.ddd:rr
2     ENTER^        ; sss.ddd:rr
3     INT           ; sss
4     LASTX         ; sss.ddd:rr
5     FRC           ; 0,ddd:rr
6     E3
7     *             ; ddd.rr
8     INT           ; ddd
9     E
10    +             ; ddd+1
11    X<>Y          ; sss
12    -             ; nn=(ddd-sss)+1
13    E5
14    /             ; 0,000nn
15    X<>Y          ; sss.ddd:rr
16    FIX 3
17    RND           ; sss.ddd
18    FIX 5
19    E-3
20    +
21    +             ; sss,(ddd+1):nn
22    XRGMOV
23    X<>Y
24    RTN
```

```
25    LBL "XM^2"    sss.ddd:rr
26    XMSQ?
27    GTO 00
28    "NOT SQUARE"
29    PROMPT
30    RTN
31    LBL 00
32    XROM "XM="    ; make a copy
33    ENTER^        ; sss.ddd:rr
34    ENTER^
35    FRC           ; 0,ddd:rr
36    FIX 3
37    RND           ; 0,ddd
38    FIX 5
39    +             ; sss,(2*ddd):rr
40    E
41    +             ; (ddd+1),(2*ddd): rr
42    ENTER^
43    FRC           ; 0,(2*ddd): rr
44    E3
45    *             ; (2*ddd),rr
46    INT           ; 2*ddd
47    E
48    +             ; 1+2*ddd
49    M*M
50    END
```

Linear Systems.

"XLS3" allows you to solve linear systems, including overdetermined and underdetermined systems.
You can also invert up to a 12x12 matrix. Its objective is to reduce the matrix on the upper left to a diagonal form with only ones in the diagonal.
The determinant of this matrix is also computed and stored in register R00.
(if there are more rows than columns, R00 is not always equal to the determinant of the upper left matrix because of row exchanges)

- If flag F01 is clear, Gaussian elimination with partial pivoting is used.
- If flag F01 is set, the pivots are the successive elements of the diagonal. This can sometimes be useful for matrices like Pascal matrices of high order. They are extremely troublesome and many roundoff errors can occur. But if you set flag F01, all the coefficients will be computed with no roundoff error at all, because all the pivots will be ones!

One advantage of this program is that you can choose the beginning data register - except R00 - (this feature is used to solve non-linear systems too):

1. You store the first coefficient into Rbb , ... , up to the last one into Ree (column by column) (with bb > 00)
2. Then you key in bbb.eeerr ENTER^, n, ENTER^, m , XEQ "XLS3" and the system will be solved. (bbb.eeerr ends up in L-register)

where r is the number of rows of the matrix and m, n are the number of rows and columns of the combined matrix including the independent terms *in the FIRST column.*

In this variant, the matrix is the right-part of the array so that, when the program stops the solution (x1 , x2 , ... , xn) is in registers R01 R02 ..... Rnn

Here, the attempt to diagonalization starts by the lower right corner of the matrix.
Flags F00 & F01 play the same role as above.

(SIZE n.m+1)

STACK	INPUT	OUTPUT
T	/	n.nnn
Z	/	/
Y	n	/
X	m	det A

n = number of rows
m = number of columns

T-output is useful to retrieve n

Don't interrupt "XLS3" because registers P and Q are used (there is no risk of crash, but their contents could be altered)

Example:

2x + 3y + 5z + 4t = 39		39 = 2x + 3y + 5z + 4t
-4x + 2y + z + 3t = 15	is re-written	15 = -4x + 2y + z + 3t
3x - y + 2z + 3t = 19		19 = 3x - y + 2z + 3t
5x + 7y - 3z + 2t = 18		18 = 5x + 7y - 3z + 2t

and we store these 20 numbers:

39	2	3	5	4		R01	R05	R09	R13	R17	
15	-4	2	1	3	in	R02	R06	R10	R14	R18	respectively
19	3	-1	2	3		R03	R07	R11	R15	R19	
18	5	7	-3	2		R04	R08	R12	R16	R20	

There are 4 rows and 5 columns,

```
CF 00   CF 01
4  ENTER^
5  XEQ "LS3"  >>>>  Det A = 840 = R00
```

Registers R05 thru R16 now contain the unit matrix and registers R01 thru R04 contain the solution x = 1 , y = 2 , z = 3 , t = 4

Thus, the array has been changed into:

1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

the solution is the first column of the new matrix.

-When the program stops, R00 = det A
-If you have to invert a matrix, the inverse will be in registers R01 thru Rn2 at the end

Determinant of order n

"XDET" simply uses "LS3" to compute the determinant of a square matrix of order n.

Flags: CF 00 = a pivot p is regarded as zero if | p | < 10-7 ; CF 01 = partial pivoting
SF 00 = a pivot p is regarded as zero if p = 0 ; SF 01 = no pivoting

STACK	INPUT	OUTPUT
X	n	Determinant

where n is the order of the square matrix.

Example: Calculate

```
            | 4  9  2 |          R01  R04  R07
      D  =  | 3  5  7 |   into   R02  R05  R08    respectively
            | 8  1  6 |          R03  R06  R09
```

3 XEQ "DET" =>> Det = 360

Example. Calculate the determinant of anti-Identity matrix of orders 10, 20 and 24.

Let's define an anti-Identity matrix as that with all elements equal to one, except the diagonal which has zeroes. For example, the 900-element, 30x30 matrix below is said to be anti-Identity:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
21	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
23	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
25	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
27	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
29	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

The first step to this assignment is to create the matrices. Obviously entering the elements by hand is not a sensible choice (up to 900 elements by hand!), so we'll first write a couple of short routines for this kind of data entering.

YONE	creates an all-ones square matrix.	Input bbb.eeerr in X
YZDG	creates a zero-diagonal matrix.	Input bbb.eeerr in X
YIDN	creates an identity matrix,	Input bbb.nnnrr in X

Equipped with these tools it's a trivial matter to create the matrices for the example:

1,10010, XEQ "XONE"	=>	a 10x10 matrix with all elements equal to "1"
RCL Z, XEQ "XZDG"	=>	an anti-diagonal 10x10 matrix
CF 00, 10, XEQ "XDET"	=>	final result

The results and execution times (at TURBOx50) are given below. Note how the accuracy holds even for very large systems – although the execution time is somewhat longer than ideal.

N	Input	Det	Time
10	1,10010	-9.000000002	35"
20	1,40020	-19.00000000	~2 min 30"
30	1,90030	-29.99999977	~11 min

Routines Listing.

```
01  LBL "XONE"
02  ENTER^
03  FIX 3
04  RND
05  E
06  XEQ 00
07  RTN
08  RCL Z
09  LBL "XZDG"
10  E-5
11  +
12  0
13  XEQ 00
14   RTN
```

```
15  LBL "XIDN"
16  ENTER^
17  FIX 3
18  RND
19  CLXRGX
20  X<>Y
21  E-5
22  +
23  E
24  LBL 00
25  XSTO IND Y   (3074)
26  ISG Y
27  GTO 00
28  FIX 6
29  END
```

Note that **XIDN** is not required for the example, but it's a symmetrical application of the same technique – whilst also showing a neat trick with the **CLYRGX** function.

Quite clearly the success of this operation is to be attributed to the **YDET** program – a straight-forward adaptation of the DET routine written by JM Baillard. The program is listed in next page, but you should refer to the original documentation available at the URL below:

http://hp41programs.yolasite.com/determinant.php

Program listing.-

```
01 *LBL "XDET"
02  ENTER^
03 *LBL "XLS3"
04  RUNNING
05  X<>Y
06  0.1
07  %
08  +
09  STO N
10  ST* Y
11  FRC
12  -
13  STO O
14   E
15  STO 00
16  LASTX
17  %
18  RCL Z
19  INT
20  +
21 *LBL 01
22  STO M
23  FS? 01
24  GTO 04
25 INT
26 RCL O
27 FRC
28 +
29 ENTER^
30 ENTER^
31 CLX
32 *LBL 02
33  XRCL IND Z  (3073}
34  ABS
35  X>Y?
36  STO Z
37  X>Y?
38  +
39  RDN
40  DSE Z
41  GTO 02
42  RCL M
43  ENTER^
44  FRC
45  R^
46  INT
47  +
48  X=Y?
49  GTO 04
50 *LBL 03
51  XRCL IND X  (3075)
52  XX<> IND Z  (3073)
53  XSTO IND Y  (3074)
54  DSE Y
55  RDN
56  DSE Y
57  GTO 03
58  RCL 00
59  CHS
60  STO 00
61 *LBL 04
62  CLX
63  FC? 00
64   E-7
65  XRCL IND M  (3077)
66  ST* 00
67  ABS
68  X<=Y?
69  CLX
70  X=0?
71  STO 00
72  X=0?
73  GTO 09
74  RCL M
75  LASTX
76 *LBL 05
77  XST/ IND Y  (3074)
78  DSE Y
79  GTO 05
80  RCL O
81  STO  P
82 *LBL 06
83  RCL  M
84  ENTER^
85  FRC
86  RCL  P
87  INT
88  +
89  X=Y?
90  GTO 08
91  XRCL IND X  (3075)
92  SIGN
93  RDN
94 *LBL 07
95  XRCL IND Y  (3074)
96  LASTX
97  *
98  XST- IND Y  (3074)
99  DSE Y
100  RDN
101  DSE Y
102  GTO 07
103 *LBL 08
104  DSE  P
105  GTO 06
106 *LBL 09
107  RCL N
108  ST-  O
109  RCL  O
110  RCL  M
111   E
112  -
113  X<=Y?
114  CLX
115  DSE X
116  GTO 01
117  RCL 00
118  CLD
119  CLA
120  END
```