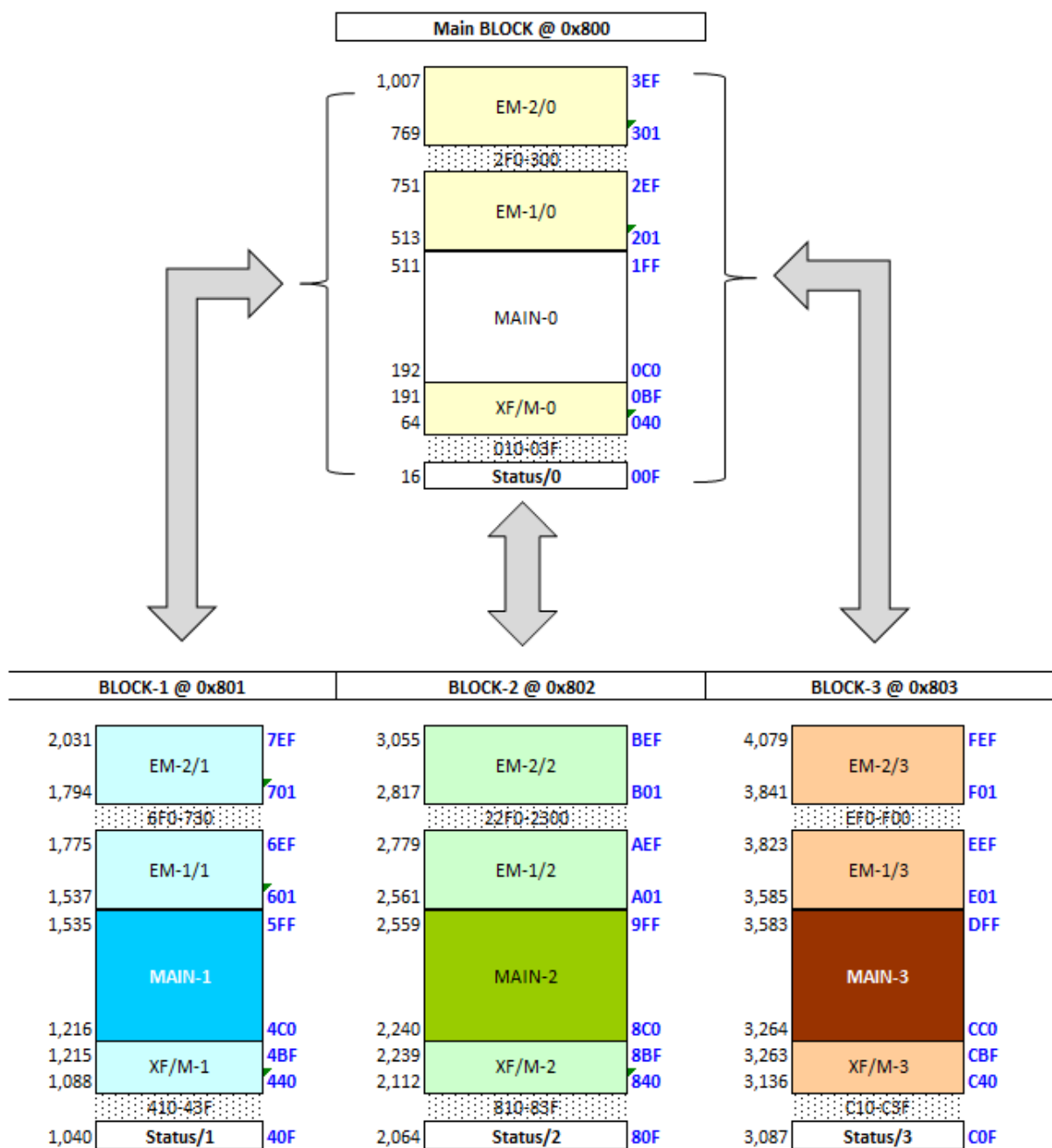# 4 ICL EXPANDED_MEMORY - Utilities Module
With Register Management and Stack Swap Functions
Revision 2C

Written and programmed by Ángel Martin
June 19, 2019

This compilation revision 2.2.2

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

Module Main Function Table and Descriptions.

#	Function	Description	Dependency	Type	Author
0	-CLXMEM 2C	*Shows "RUNNING" msg*	Lib#4	MCODE	*Ángel Martin*
1	*"BK-MM"*	Block to Main memory	YFNX; OSX3	FOCAL	*Ángel Martin*
2	*"BK-XM"*	Block to Extended memory	YFNX; OSX3	FOCAL	*Ángel Martin*
3	*"BKRST"*	Alternate Block Reset (Clear)	YFNX; OSX3	FOCAL	*Ángel Martin*
4	*"BKSWP"*	Alternate Block Swap	YFNX; OSX3	FOCAL	*Ángel Martin*
5	*"MM-BK"*	Main memory to Block	YFNX; OSX3	FOCAL	*Ángel Martin*
6	*"MMRCL"*	Main Memory Recall	OSX3	FOCAL	*Ángel Martin*
7	*"MMSTO"*	Main memory Storage	OSX3	FOCAL	*Ángel Martin*
8	*"MMSWP"*	Main Memory Exchange	OSX3	FOCAL	*Ángel Martin*
9	*"OKM"*	Validates MM set	Lib#4, YFNX	FOCAL	*Ángel Martin*
10	*"OKM?"*	Checks for valid MM set	Lib#4, YFNX	FOCAL	*Ángel Martin*
11	*"OKX"*	Validates XM set	Lib#4, YFNX	FOCAL	*Ángel Martin*
12	*"OKX?"*	Checks for valid XM set	Lib#4, YFNX	FOCAL	*Ángel Martin*
13	*"RAMRCL"*	Complete RAM Recall	YFNX; OSX3	FOCAL	*Ángel Martin*
14	*"RAMSTO"*	Complete RAM Storage	YFNX; OSX3	FOCAL	*Ángel Martin*
15	*"RAMSWP"*	Complete RAM Exchange	YFNX; OSX3	FOCAL	*Ángel Martin*
16	*"XM-BK"*	Extended memory to Block	YFNX; OSX3	FOCAL	*Ángel Martin*
17	*"XMRCL"*	Extended memory Recall	OSX3	FOCAL	*Ángel Martin*
18	*"XMSTO"*	Extended memory Storage	OSX3	FOCAL	*Ángel Martin*
19	*"XMSWP"*	Extended memory Exchange	OSX3	FOCAL	*Ángel Martin*
20	-YFNM BCKUP	*Lib#4 Check & Splash*	Lib#4	MCODE	*Nelson F. Crowle*
21	MM-YBK	Main memory to Block	Lib#4	MCODE	*Martin-Dalrymple*
22	MMYSWP	Main Memory Exchange	Lib#4	MCODE	*Martin-Dalrymple*
23	ST-YBK	Status Regs to Block	Lib#4	MCODE	*Martin-Dalrymple*
24	STYSWP	Status Regs exchange	Lib#4	MCODE	*Martin-Dalrymple*
25	XM-YBK	Extended memory to Block	Lib#4	MCODE	*Martin-Dalrymple*
26	XMYWP	Extended memory Exchange	Lib#4	MCODE	*Martin-Dalrymple*
27	YBK-MM	Block to Main memory	Lib#4	MCODE	*Martin-Dalrymple*
28	YBK-ST	Block to Status Regs	Lib#4	MCODE	*Martin-Dalrymple*
29	YBK-XM	Block to Extended memory	Lib#4	MCODE	*Martin-Dalrymple*
30	YBKSWP	Alternate Block Swap	Lib#4	MCODE	*Martin-Dalrymple*
31	YEDIT	Edit/View Block	Lib#4, YFNX, OSX3	FOCAL	*Ángel Martin*
32	YINPT _	Input control string	Lib#4	MCODE	*Ángel Martin*
33	YMEM	Memory Functions Launcher	Lib#4	MCODE	*Ángel Martin*
34	YMMOVE	General-Purpose Move	Lib#4	MCODE	*Monte Dalrymple*
35	YMSWAP	General-Purpose Swap	Lib#4	MCODE	*Monte Dalrymple*
36	-STKSWP	Stack Swap Launcher	Lib#4	MCODE	*Ángel Martin*
37	?RAM	Checks if running from RAM	Lib#4	MCODE	*Ángel Martin*
38	A<>YRG _ _ _	Sawps ALPHA and Y-Regs	Lib#4	MCODE	*Ángel Martin*
39	CLYRG	Clears All Expanded Regs	Lib#4	MCODE	*Ángel Martin*
40	CLYRGX	Clears Y-Regs by X	Lib#4	MCODE	*Ángel Martin*
41	CPYBNK _"_:_	Copies bank-switched page	Lib#4	MCODE	*Ángel Martin*
42	ST<>YRG _ _ _	Swaps Stack and Y-Regs	Lib#4	MCODE	*Ángel Martin*
43	YF$ _	Sub-function Launcher by Name	Lib#4	MCODE	*Ángel Martin*
44	YF# _ _ _	Sub-function Launcher by index	Lib#4	MCODE	*Ángel Martin*
45	YRGMOV	Block-moves Y-Regs	Lib#4	MCODE	*Ángel Martin*
46	YRGSWP	Block-Swaps Y-Regs	Lib#4	MCODE	*Ángel Martin*
47	-Y-REG FNS	Section Header	n/a	MCODE	*Ángel Martin*
48	YARC _ _ _	Expanded Reg. ARCL	Lib#4	MCODE	*Ángel Martin*
49	YAST _ _ _	Expanded Reg. ASTO	Lib#4	MCODE	*Ángel Martin*
50	YDSE _ _ _	Expanded Reg DSE operation	Lib#4	MCODE	*Ángel Martin*
51	YISG _ _ _	Expanded Reg ISG operation	Lib#4	MCODE	*Ángel Martin*
52	XRCL _ _ _	Expanded Reg. Recall	Lib#4	MCODE	*Ángel Martin*
53	YRC+ _ _ _	YRCL Addition	Lib#4	MCODE	*Ángel Martin*

54	YRC- _ _ _	YRCL Subtract	Lib#4	MCODE	Ángel Martin
55	YRC* _ _ _	YRCL Multiply	Lib#4	MCODE	Ángel Martin
56	YRC/ _ _ _	YRCL Division	Lib#4	MCODE	Ángel Martin
57	YSTO _ _ _	Expanded Reg Store	Lib#4	MCODE	Ángel Martin
58	YST+ _ _ _	YSTO Addition	Lib#4	MCODE	Ángel Martin
59	YST- _ _ _	YSTO Subtract	Lib#4	MCODE	Ángel Martin
60	YST* _ _ _	YSTO Multiply	Lib#4	MCODE	Ángel Martin
61	YST/ _ _ _	YSTO Division	Lib#4	MCODE	Ángel Martin
62	YVEW _ _ _	View Expanded Register	Lib#4	MCODE	Ángel Martin
63	YX<> _ _ _	Expanded Reg Exchange	Lib#4	MCODE	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	?LIB4	Checks for Library#4	Lib#4	MCODE	Ángel Martin
17	?YFNX	Checks for YFNX ROM	Lib#4	MCODE	Ángel Martin
18	ASWP>	Swaps ALPHA around '>"	Lib#4	MCODE	W&W GmbH
19	D>H	Decimal to Hex	Lib#4	MCODE	William Graham
20	H>D	Hex to Decimal	Lib#4	MCODE	William Graham
21	YFINDX	Finds register with X-value	Lib#4	MCODE	Ángel Martin
22	FCAT	Sub-function Catalog	Lib#4	MCODE	Ángel Martin

Note that the new Expanded Register Management functions have completely replaced the page management from previous versions of the CLXMem module. These continue to be available in the PowerCL_Extreme, under the sub-functions FATs.

Introduction – Unleashing the CL RAM

This module includes utility functions to take advantage of the CL RAM in a very convenient way, modeled after the standard (native) utilization of the basic calculator RAM. Anyone already familiar with the HP-41C data registers and stack operation will have an immediate and intuitive grasp of the expanded registers functions – as they simple extend the same model to the CL RAM.

There are three general groups of functions and programs, represented by the three sections in the main FAT as described below.

- The module starts with a set of FOCAL programs used to make bulk back-ups of the calculator RAM in the CL RAM. These programs are a good vehicle for getting an understanding of the basic CL memory arrangement, and how it becomes accessible using some advanced functions included in the YFNX and OS/X modules.

- The second section is a set of MCODE functions also related to the calculator RAM backup to and restore from CL RAM. These functions can replace the FOCAL programs in a more efficient way – faster and somehow more flexible, with equivalent results even if they don't have entirely identical approaches.

- The third section introduces the MCODE functions used to access individual Extended registers within the CL RAM. A total of 1,024 registers are accessible using them, and the function set is very complete – even adds RCL math, which is lacking in the mainframe OS. The user interface mimics that of the mainframe functions, extended with novel use cases like using standard registers for indirect addresses of expanded registers.

Module Dependencies.

The CLXMEM 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 most of the FOCAL programs from the first section. This module provides advanced OS extensions and therefore it's recommended to have it always plugged in the machine – a real power user can't live without it.

An obvious dependency is the CL itself: all MCODE functions will check for it, showing and error message and aborting the execution if they're running on standard HP-41's or SW emulators like V41.
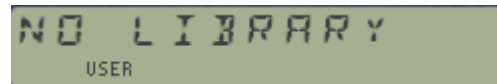
Finally, and like the PowerCL Extreme, the CLXMEM is also meant to be used paired with the YFNS Extreme Module, i.e. YFNX. Note however that the PowerCL Extreme is not required to use the CLXMEM functions.

> *Note: The CL expanded Registers module requires the Library#4 revision R4 or higher plugged in. It's also meant to be paired with the YFNS Extreme module, YFNX.*

Managing the CL Expanded Y-registers

The CL board has three 4k-blocks of RAM memory reserved for extensions to the OS. So far these have been rather ignored by all applications, but finally they're put to a good use with the function sets included in this module.

This first section covers the individual access of the expanded Y-registers included in the RAM block located at 0x801, i.e. the first of those three RAM blocks. You'll be able to store, recall, view, exchange, and perform ISG/DSE operations on 1,024 of those registers as if they were standard 41 registers within main memory. Note the presence of the arithmetic operations as well.
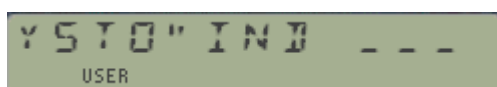
Extended Regs	Store	Recall	Other	X-Blocks
Y-Register	YSTO _ _ _	YRCL _ _ _	YX<> _ _ _	CLYRG
From 0 to 1,023.	YST+ _ _ _	YRC+ _ _ _	YVEW _ _ _	CLYRGX
Use EEX for the	YST- _ _ _	YRC- _ _ _	YDSE _ _ _	YRGMOV
Fourth digit field.	YST* _ _ _	YRC* _ _ _	YISG _ _ _	YRGSWP
	YST/ _ _ _	YRC/ _ _ _	YFINDX	ST<>YRG _ _ _
ALPHA	YAST _ _ _	YARC _ _ _		A<>YRG _ _ _

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.
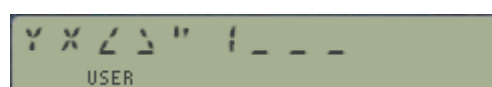
Most of the functions will prompt for the parameters to use. The initial prompt is a three-field underscore for the Y-register indeed. Pressing [SHIFT] changes it to IND three digit fields for another Y-register to be used as indirect. Pressing the [RADIX] key changes to the IND ST _ prompt, where you'll 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.
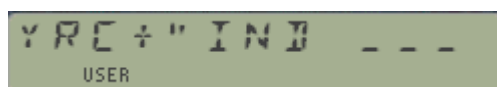
You can move between the functions while the prompts are up; not only to select the math operation but also to change the main function amongst the group. So for instance during the YRCL _ _ _ main prompt pressing the ⌗SST⌗ key will trigger the **YX<>** function, or pressing ⌗STO⌗ will invoke the **YSTO** 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 1,023 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 **YRCL** function:

Although possible, it is however not meant to be used in a program because of the obviously higher byte count. You should use the native STO/RCL functions instead for direct stack and data registers in programs – which has the additional benefit of a clearer representation by the OS as merged lines.

Storing and Recalling ALPHA Data

The expanded functions **YAST** and **YARC** 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 YSTO/YRCL prompts by pressing the ALPHA key at any time.

Deleting Expanded Registers.

The function **CLYRG** will delete all the 1,024 expanded registers, and therefore it's equivalent to **YMCLR** used on the entire 0x801 block, as follows: "801000-0FFF", **YMCLR**
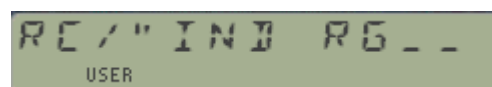
Additionally with **CLYRGX** you can selectively delete a defined block of expanded registers as defined by its control word (in X) "bbb.eeeiii", where "bbb.eee" defines the beginning and end of the registers range to delete, and "iii" the increment between them. For contiguous registers "zero" is equivalent to just "one".

Other Block Operations.

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

- **A<>YRG** and **ST<>YRG** exchange a group of five expanded 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.

- **YRGMOV** and **YRGSWP** can be used to move or exchange a block of expanded registers at once – either contiguous or in an increment pattern as provided by the control word in the X-register. Much the same as the X-Functions RGMOVE and RGSWAP - in case you wonder.

Moving around the Expanded Registers Functions.
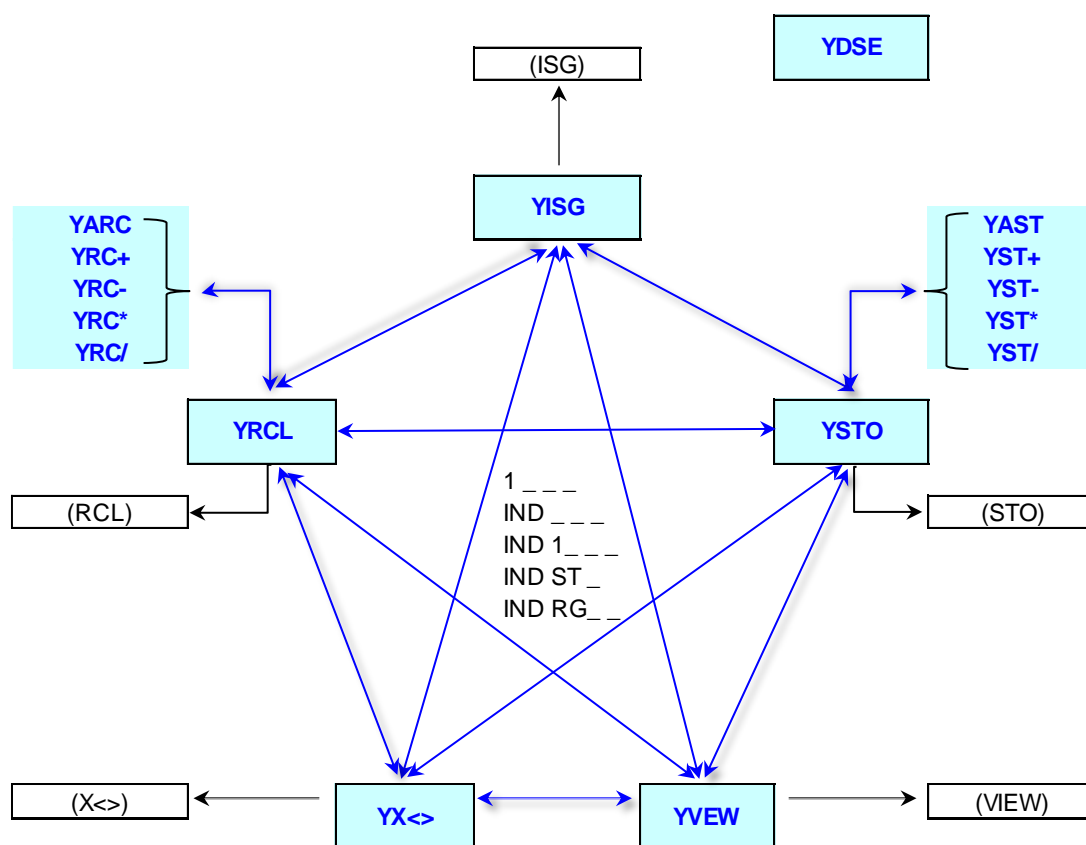
Figure 1.

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 **YSTO** to the STO key.

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

- RCL will toggle to YRCL _ _ _
- SST (for X<>) will launch YX<> _ _ _
- R/S (for VIEW) will launch YVEW _ _ _
- CHS (for ISG) will launch YISG _ _ _
- ALPHA will trigger YAST _ _ _
- The math keys will launch the corresponding math function, ie. YST+ _ _ _
- EEX will add one field to the prompt: YSTO 1 _ _ _
- SHIFT will add the IND prompt (assuming EEX isn't already up)
- 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 15 functions (**YDSE** is not part of this scheme) – regardless of which one you used to start the sequence. See the descriptions earlier in this manual for additional details.

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

For those times when you'd like to know if a certain value is stored in the Y-data register, the sub-function **YFINDX** (a.k.a. **XF# 21**) is available to do a cursory comparison looking for a match with the value in the X-register. All Y-data registers are checked, starting with YR00 until YR1023 – which could take a long time depending on where the match exists.

The function returns the number of the first Y-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 YR999, as well as an equivalent routine for the standard data registers - for comparison purposes. See the Total_Rekall manual for yet another routine to tackle this "where is Waldo" problem using other advanced functions.

01	LBL "YFNDX"		01	LBL "FINDX"
02	.999		02	SIZE?
03	X<>Y		03	E
04	LBL 00		04	–
05	**YRCL (IND Y)**		05	E3
06	**3074**		06	/
07	X=Y?		07	X<>Y
08	GTO 02		08	LBL 00
09	RDN		09	RCL IND Y
10	ISG X		10	X=Y?
11	GTO 00		11	GTO 02
12	CLX		12	RDN
13	-1		13	ISG Y
14	RTN		14	GTO 00
15	LBL 02		15	CLX
16	X<> Z		16	-1
17	INT		17	RTN
18	END		18	LBL 02
			19	X<> Z
			20	INT
			21	END

The possibilities of having an additional set of 1,024 registers available to your own programs are wide and deep. For starters you could permanently operate with a SIZE 000 and use all the 319 standard registers 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. As of release 2B even the ALPHA storage functions **YAST** and **YARC** are included, which can also use the expanded register range.

Then you have the benefit of a much larger set of registers 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 1,000 registers in a very efficient way. – with random data populating those registers it took about 32 seconds to sort 1,000 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 "YVIEW"
02	LBL 00
03	"YR"
04	ARCLI
05	"|-: "
06	**YRCL (IND X)**
07	**3075**
08	ARCLX
09	AVIEW
10	PSE
11	RDN
12	ISG X
13	GTO 00
14	END

01	LBL "YRAN"
02	RCL X
03	LBL 00
04	RNDM
05	**YSTO (IND Y)**
06	**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 correspond to the following: (with SIZE 319 for the standard registers case)

1. The register index for direct access, from 0 to 1,023   [000 – 3FF]
2. The indirect register index for IND from 1,024 to 2,047   [400 – 7FF]
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 voids 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 functions MCODE, and therefore should not be used in your FOCAL programs as synthetic register when the expanded registers functions are also used.

> *Warning: The expanded Register functions are not to be used if the first block is already used as back-up location. Use blocks #2 and #3 instead!*

More application programs: the YREGAPPS Module

The YREGAPPS ROM contains a selection of Matrix and Registers applications programs mostly taken from the PPC ROM and Jean-Marc Baillard's collection. I have modified them to take advantage of the expanded registers, replacing all operation form 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…

The Matrix routines are described in Jean-Marc's excellent web site, a vast resource for HP-41 math software. The routines include three MCODE functions for matrix product, norm and trace calculation operating on the expanded registers directly – For JBM's matrix documentation see:

http://hp41programs.yolasite.com/matrixop.php,
http://hp41programs.yolasite.com/matrixmcode.php

I 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}

- **YCRYPT** encrypts the contents of a given page (obviously mapped to RAM), from the address provided in the first prompt to the end of the page, and using the encryption code provided in the second prompt. You can undo the encryption using the same values of address and code again (which therefore need to be remembered)

- **YDUMP** copies all standard registers into the Y-Reg area, using the same indexes.

- **YSHFT** does a selective copy, using the control word bbb.eee in X. Note that if this is negative then the direction of the copy is reversed, i.e. will go from the Y-Area to the standard registers. In this case, make sure the SIZE is set so that there are enough standard registers to receive the Y-data!

- **YINP** and **YOUT** are input/output routines to enter or visualize the values respectively. Require the control word bbb.eee in X. (Note: this is not the same as **YINPT**, the CL-input string function from the PowerCL or the CLXPREGS module)

- **YRAN** populates a block of Y-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.

- **YSORT** will do a descending data sort on the Y-registers block defined by the control word bbb.eee in X. This is a slow program; use the PPC versions **YS2** and **YS3** for speed. Numeric values only!

- **YM>RM** and **RM>YM**. The matrix convention used in Jean-Marc's programs stores the elements in COLUMN order, and uses the control word "**bbbeeerr**" to define the matrix, where **bbb**/**eee** are the beginning and ending registers, and "**rr**" is the number of rows. This is the "transposed" way of the method used by the Advantage and SandMatrix modules, but these two routines will move a matrix between the Y-Registers and the Standard registers, whilst performing the register transposition at the same time – In addition to moving data, you can use these also to validate your results - but remember to set the SIZE large enough for the matrix to fit in the standard registers area. For matrices in X-Mem you can use the **MMOVE** function in the Advantage to move them first to the standard registers.

The following lists the programs contained in the YREGAPPS Module:

#	Function	Description	Dependency	Type	Author
021.00	-Y-RGS_1B	Shows 'RUNNING...' msg.	Lib#4	MCode	Ángel Martin
021.01	STVIEW	Stack View	Lib#4	MCODE	Ángel Martin
021.02	YCRYPT	Encrypts Page	CL, OSX	Hybrid	Ángel Martin
021.03	"YDUMP	Dumps Standard Regs into Y-	CLXPRGS, OSX	FOCAL	Ángel Martin
021.04	"YINP	Inputs Y-Regs	CLXPRGS, OSX	FOCAL	Ángel Martin
021.05	"YOUT	Outputs Y-Regs	CLXPRGS, OSX	FOCAL	Ángel Martin
021.06	"YRAN	Enters Random values	CLXPRGS, OSX	FOCAL	Ángel Martin
021.07	"YSHFT	Selective Std, Reg copy	CLXPRGS, OSX	FOCAL	Ángel Martin
021.08	"YSORT	Sorts Y-Regs	CLXPRGS	FOCAL	JM Baillard
021.09	"YM>RM (*)	Moves matrix to Standard Regs	CLXPRGS, ADVTG	FOCAL	Ángel Martin
021.10	"RM>YM (*)	Moves matrix to Y-Regs	CLXPRGS, ADVTG	FOCAL	Ángel Martin
021.11	-YPPC_MTRX	Section Header	n/a	MCODE	Ángel Martin
021.12	"QR	Quotient / Remainder	none	FOCAL	PPC
021.13	"YMIO	Matrix I/O	CLXREGS	FOCAL	PPC
021.14	"YRRM	Row Reduction Matrix	CLXPRGS	FOCAL	PPC
021.15	"YM1	Interchange two rows	CLXPRGS	FOCAL	PPC
021.16	"YM2	Multiply row by constant	CLXPRGS	FOCAL	PPC
021.17	"YM3	Add multiple of row to another	CLXPRGS	FOCAL	PPC
021.18	"YM4	Register address to (i, j)	CLXPRGS	FOCAL	PPC
021.19	"YM5	(i, j) to Register address	CLXPRGS	FOCAL	PPC
021.20	"YS1	Stack Sort	CLXPRGS	FOCAL	PPC
021.21	"YS2	Small Register Sort	CLXPRGS	FOCAL	PPC
021.22	"YS3	Large Register Sort	CLXPRGS	FOCAL	PPC
021.23	-YPPC_BLCK	Section Header	n/a	MCODE	Ángel Martin
021.24	"YBC	Block Clear	CLXPRGS	FOCAL	PPC
021.25	"YBE	Block Exchange	CLXPRGS	FOCAL	PPC
021.26	"YBI	Block Increment	CLXPRGS	FOCAL	PPC
021.27	"YBM	Block Move	CLXPRGS	FOCAL	PPC
021.28	"YBR	Block Rotate	CLXPRGS	FOCAL	PPC
021.29	"YBV	Block View	CLXPRGS	FOCAL	PPC
021.30	"YBX	Block Extrema	CLXPRGS	FOCAL	PPC
021.31	"YB?	Block Statistics	CLXPRGS	FOCAL	PPC
021.32	"YDR	Delete Record	CLXPRGS	FOCAL	PPC
021.33	"YIR	Insert Record	CLXPRGS	FOCAL	PPC
021.34	"YMS	Memory to Stack	CLXPRGS	FOCAL	PPC
021.35	"YPR	Pack Registers	CLXPRGS	FOCAL	PPC
021.36	"YSM	Stack to Memory	CLXPRGS	FOCAL	PPC
021.37	"YUR	Unpack Registers	CLXPRGS	FOCAL	PPC
021.38	-YJMB_MTRX	Section Header	n/a	MCODE	Ángel Martin
021.39	"CRYMAT	Create Matrix from function	CLXPRGS	FOCAL	JM Baillard
021.40	M*M	Matrix Product	CL, Lib#4	MCODE	JM Baillard
021.41	MNORM	Matrix Norm	CL, Lib#4	MCODE	JM Baillard
021.42	TRACE	Matrix Trace	CL, Lib#4	MCODE	JM Baillard
021.43	"YDET	Determinant	CLXPRGS	FOCAL	JM Baillard
021.44	"YLS	Linear Systems	CLXPRGS	FOCAL	JM Baillard
021.45	"YLS1	Linear Systems (variant)	CLXPRGS	FOCAL	JM Baillard
021.46	"YM-	Matrix Subtraction	CLXPRGS	FOCAL	JM Baillard
021.47	"YM+	Matrix Addition	CLXPRGS	FOCAL	JM Baillard
021.48	"YM*	Matrix Product	CLXPRGS	FOCAL	JM Baillard
021.49	"YMCO	Matrix Copy	CLXPRGS	FOCAL	JM Baillard
021.50	"YMINV	Inverse Matrix	CLXPRGS	FOCAL	JM Baillard

021.51	**"YMNORM**	Matrix Norm	CLXPRGS	FOCAL	*JM Baillard*
021.52	**"YMPL**	Matrix Polynomial	CLXPRGS	FOCAL	*JM Baillard*
021.53	**"YMRCL (*)**	Recall Matrix	CLXPRGS, OSX	FOCAL	*Ángel Martin*
021.54	**"YMSTO (*)**	Store Matrix	CLXPRGS, OSX	FOCAL	*Ángel Martin*
021.55	**"YRANM**	Random (integers) Matrix	CLXPRGS, TIMER	FOCAL	*JM Baillard*
021.56	**"YRNSYM**	Symmetric RANM	CLXPRGS, TIMER	FOCAL	*JM Baillard*
021.57	**"YTRACE**	Matrix Trace	CLXPRGS	FOCAL	*JM Baillard*
021.58	**"YTRN**	Transpose Matrix	CLXPRGS	FOCAL	*JM Baillard*
021.59	**"YTRN2**	Symmetric TRN	CLXPRGS	FOCAL	*JM Baillard*

(*) See the program listings in the appendices.

Figure 2. – Conceptual scheme of the different CL sRAM blocks.
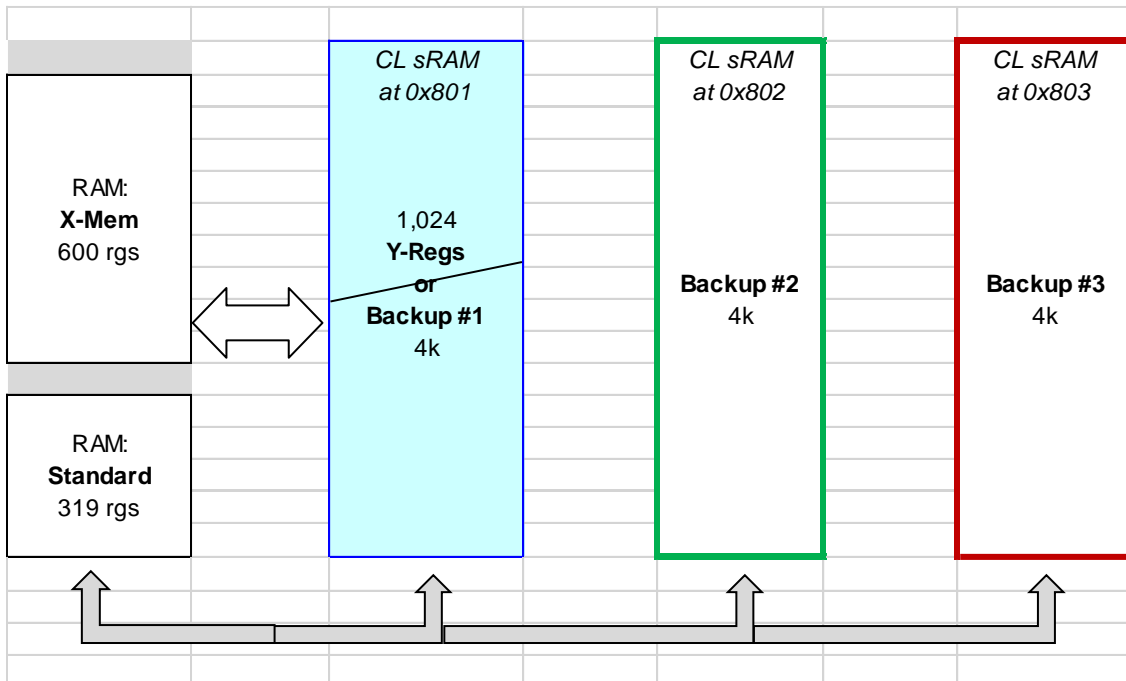
> *Warning: The expanded Register functions are not to be used if the first block is already used as back-up location. Use blocks #2 and #3 instead!*

Example. Calculate the determinant of anti-Identity matrix of orders 10, 20 and 30.

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 "YONE"	=>	a 10x10 matrix with all elements equal to "1"
RCL Z, XEQ "YZDG"	=>	an anti-diagonal 10x10 matrix
CF 00, 10, XEQ "YDET"	=>	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 "YONE"**	16	ENTER^
02	ENTER^	17	FIX 3
03	FIX 3	18	RND
04	RND	**19**	**CLYRGX**
05	E	20	X<>Y
06	XEQ 00	21	E-5
07	RTN	22	+
08	RCL Z	23	E
09	**LBL "YZDG"**	24	LBL 00
10	E-5	**25**	**YSTO IND Y** (3074)
11	+	26	ISG Y
12	0	27	GTO 00
13	XEQ 00	28	FIX 6
14	RTN	29	END
15	**LBL "YIDN"**		

Note that **YIDN** 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. You should refer to the original documentation available at the URL below:

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

Appendix 0.- Program Listings: Moving Matrices

Note that when copying matrices between register areas, it is the transposed matrix what gets moved. If the original matrix is in the Y-Regs, say with control word bbb.eeerr1, then its transposed is created first in a group of contiguous registers, starting at eee+1. Then it is copied to the standard registers area using **YSHFT**, and a matrix is "defined" using the "R(bbb+1)" name as per the Advantage functionality. You can move it back to Y-Regs after this is complete, effectively undoing the initial transposition but now stored in the same location the transposed matrix was created, starting at eee+1.

However, the opposite direction has a different (and simpler) behavior: if you start moving a matrix "Rn" from the standard registers area its transpose will replace the original one first, and then it will be moved to the Y-Regs starting at register nnn. The routines are shown below.

1	**LBL "YM>RM"**	bbb'.eee'rr'		36	*****	
2	**ENTER^**			37	**INT**	
3	**FRC**			38	**E3**	
4	**E3**			39	**/**	*removes the "rr"*
5	*****	beee'.rr'		40	**CHS**	*from Y-Regs to Standard*
6	**FRC**	0.rr'		41	**XROM "YSHFT"**	*copy data over*
7	**E2**			42	**RCL 00**	
8	*****	rr'		43	**MATDIM**	
9	**STO 00**	0,0cc		44	**RTN**	*done.*
10	**X<>Y**			**45**	**LBL "RM>YM"**	*expects "Rn" in ALPHA*
11	**ENTER^**			46	**DIM?**	rr,0cc
12	**FRC**	bbb		47	**INT**	rr,0cc
13	**E3**			48	**E5**	
14	*****			49	**/**	0,000rr
15	**INT**			50	**STO 00**	
16	**E**			51	**TRNPS**	
17	**+**			52	**DIM?**	cc,0rr
18	**XROM "YTRN"**	bbb.eeerr		53	**INT**	cc,0rr
19	**ENTER^**			54	**LASTX**	
20	**ENTER^**			55	**FRC**	0.0rr
21	**INT**	bbb		56	**E3**	
22	**E**			57	*****	rr,0cc
23	**-**	bbb-1		58	*****	cc*cc = size
24	**"R"**			59	**ANUM**	n
25	**ARCLI**			60	**+**	n+cc*rr
26	**RDN**	bbb.eeerr		61	**E3**	
27	**FRC**	0.eeerr		62	**/**	0.0(n+cc*rr)
28	**E3**	eee.rr		63	**ANUM**	n
29	*****			64	**E**	
30	**FRC**	0.rr'		65	**+**	n+1
31	**E1**			66	**+**	(n+1),0(n+cc*rr)
32	**/**	0.0cc		67	**ST+ 00**	
33	**ST+ 00**	rr,0cc		68	**XROM "YSHFT"**	
34	**X<>Y**	bbb.eeerr		69	**RCL 00**	*brinf cnt'l word to X*
35	**E3**			70	**END**	*done.*

The Y-Register functions are within the **YSHFT** routine, used for the actual data copying.

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.

#	Instruction	Comment		#	Instruction	Comment
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.

Backing up data in the CL Expanded Memory Blocks.

The CL board has four 4k-blocks in sRAM that can be allocated for the 41 system RAM. Of these, only the first one (0x800) is utilized by the OS, holding the registers 0-3FF – that is the complete calculator memory set, including both Main and all Extended Memory.

It is therefore possible to use the other three blocks to hold backup copies of the first (default) one, or alternate sets of memory and programs. This requires a few utility functions to store, recall and exchange (swap) the block contents – which is the main subject of the CL_XMEM module.

Summary Function Table of the three scenarios:

The table below provides a comparison of the functions and routines included in the module. Note that no single set has a complete functionality, but that the MCODE scenario is by far the fastest and most convenient one to use.

Action	Memory Area	MCODE Functions	FOCAL#1 (Y-Functions)	FOCAL#2 (PEEKR/POKER)
Store	Register	**YSTO _ _ _**	-	-
	Stack	**ST-YBK**	-	-
	Main	**MM-YBK**	**MM-BK**	**MMSTO**
	Extended	**XM-YBK**	**XM-BK**	**XMSTO**
	RAM	-	**RAMSTO**	-
Recall	Register	**YRCL _ _ _**	-	-
	Stack	**YBK-ST**	-	-
	Main	**YBK-MM**	**BK-MM**	**MMRCL**
	Extended	**YBK-XM**	**BK-XM**	**XMRCL**
	RAM	-	**RAMRCL**	-
Swap	Register	**YX<> _ _ _**	-	-
	Stack	**STYSWP**	-	-
	Main	**MMYSWP**	-	**MMSWP**
	Extended	**XMYSWP**	-	**XMSWP**
	RAM	-	**RAMSWP**	-

## FOCAL Function Set #1: Using Extended PEEKR and POKER

The first set of available programs utilize the **PEEKR** and **POKER** functions in the AMC_OS/X Module, (extended versions of the originals from the CCD Module), and are summarized in the table below.

The strategy in this case is to use the expanded pages as destination of the backups; which can be done because PEEKR and POKER can operate in the full range of addresses up to FFF (hex). The advantage of this method is that the operation can either be done at the complete block level, or selectively for Main and Extended memory only. However note that there's not a function for the entire block space handled at once within this set.

F-Set-1	Store	Recall	Exchange
All	n/a	n/a	n/a
Main	**MMSTO**	**MMRCL** (*)	**MMSWP** (*)
Extended	**XMSTO**	**XMRCL**	**XMSWP**

*(*) these functions will check that the program is not running from RAM,
which would create a problem if allowed to occur for obvious reasons.*

All functions will first prompt for the alternate block# to be used in the corresponding action. The only valid answers are 1, 2, or 3 – denoting the blocks at addresses 0x400, 0x800, and 0xC00 respectively.

```
BLK _     1:2:3
USER              PRGM
```
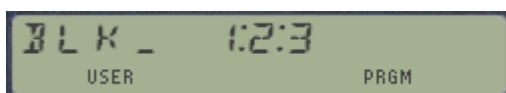
The swap action will always happen between the default block (#0) and the selected one. Info messages will pop up during the execution, indicating the memory area being transferred: "MAIN", "XF/M", "EM-1", and "EM-2". When finished the message "DONE" will be shown.

Setting flag 21 will halt the program execution just before each of the block-transfer operations. You should let the programs run until the end for a complete backup or restore action.

Refrain from using the 801 block (index "1" in the functions above) if you're already using it as expanded registers area with the functions described before.

FOCAL Function Set #2 : Using Y-Functions

A second set of programs is provided – equivalent to the ones listed above but using YFNX functions instead of PEEKR and POKER (*but the OS/X module is still needed*). The methods are equivalent but here there's a small reduction in the execution time (bulk copy as opposed to register by register) – although the feedback messages are not so self-explanatory.

Note that also in this case the backups are placed in the expanded blocks, located at addresses 0x801, 0x802, and 0x803 within the CL sRAM space.

F-Set-2	Store	Recall	Exchange
All	**RAMSTO**	**RAMRCL** (*)	**RAMSWP** (*)
Main	**MM-BK**	**BK-MM** (*)	n/a
Extended	**XM-BK**	**BK-XM**	n/a

(*) these functions will check that the program is not running from RAM,
which would create a problem if allowed to occur for obvious reasons.

Note that these programs all use the Y-Buffer block as a temporary storage location – *overwriting any previous contents of this block.* This may be specially damaging if you're using a RAM copy of the IMDB data base, so plan accordingly first!

Finally, even if they use different approaches, the back-up data sets produced by each of the function sets are interchangeable, i.e. you don't need to use the same function set for the restore and backup operations.

Swapping and Resetting alternate blocks.

Two other utility programs are available to clear and swap the contents between alternate blocks, as follows:

Utilities	Description	Input	
BKRST	Clears Block	Block number 1:2:3	
BKSWP	Swaps Blocks	FROM/TO blocks	

The prompt will only allow values {1,2,3} to be entered – thus ensuring that no accidental erasing or corruption will occur on the main block. Note however that there is no protection against using any of the restoring programs with a cleared block – which will completely erase your complete main or extended memory (or both).

Backups in sRAM will survive Memory Lost conditions, but they require the battery to be charged and in the calculator. For a permanent backup even without the batteries refer to the POWER_CL Module manual, functions **YWALL** and **YRALL**.

Data Protection: Validity tokens.

As a general rule, all backup programs will write a check token in the alternate blocks to flag them for valid data sets. Reversely, this token will be read by all the restore programs, aborting if it's not already present. This functionality resides in the following set of auxiliary functions:

Action	Main Memory	Extended Memory	
Write	**OKM**	**OKX**	
Check	**OKM?**	**OKX?**	
Location	**0x017**	**0x300**	

Storage programs will write the validity token in the destination block. No checks are made.

Recall programs will check for valid data sets at the backup block. If the Main memory token is not correct, the execution will stop with the message "MM NOT OK". This will persist upon R/S. However if the Extended memory token is the problem, the warning message "XM NOT OK" will only show once – and the user can choose to continue (pressing R/S) regardless. Be aware that the XM at the main block will probably be wiped out as a result.

Exchange programs will first check for valid data sets, and if present will brand the blocks as valid. The same considerations apply as for the recall case above.

Dependencies.

The CL_XMEM module functions are designed to use the Library-4, make sure it is installed. There's a library presence check upon the calculator ON event, showing the "NO LIBRARY" error message if not found.
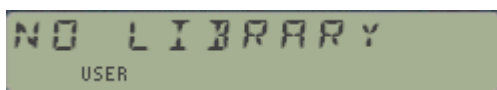
Besides the Library-4, the **YFNX** module and the **AMC_OS/X** module (CL mnemonic "OSX3") are required. A "NONEXISTENT" message will show if not found during the execution.

The expanded registers functions will check for CL hardware before performing the store action. This protects non-CL models, which default to the standard addresses and therefore could cause Memory Lost conditions if used on non-CL machines.

Appendix. Restoring to the Main block using the Set#2.

When using the YFNX functions to do the backup and restore, the only way to read the data from the Y-buffer is the **YMCPY** function. This function operates on the complete 4,096-byte block, therefore including the status registers as well. This requires that *prior to calling* **YMCPY** the PC location in the Y-buffer must be overwritten with the appropriate value - for the FOCAL program to continue running as desired after the **YMCPY** action has occurred. This is implemented with the following code:

```
16
WSIZE          sets word size
R^             puts TO: adr in X
RCL b          Program Pointer to X
DCD            decode the NNN
"/-**"         padding chars
-2             rotating factor
AROT           put to left
ASHF           discard leftmost 7 chars
ASHF           ditto again
CLX            disable stack lift
H>D            convert to decimal
42             add 42 bytes (offset)
+              in ROM addresses INCREASE!
"805030-"      pointer for first byte in b(12)
ARCLH          append as Hex number
YPOKE          return-to addr
LBL 05
"805>"         prepare string for mass-transfer
ARCL Z         contains the TO: adr ("800")
AVIEW          provide feedback
YMCPY          do the transfer
TONE 0         will return here!
"DONE"         final message
AVIEW          show to user
END            all done.
```

Obviously this is only relevant when moving data back to the main block (programs **RAMRCL**, **RAMSWP**, **BK-MM** and **BK-XM**) - the OS couldn't care less about the content of the OS registers backup in the alternate blocks.

This issue does not occur with **XMSWP** or **MMSWP** because they use PEEKR / POKER to do the data transferring on an individual register basis – thus avoiding altogether the conflictive status registers.

A bit of a nuisance to worry about that would not be necessary with an MCODE function - but an interesting remark to make nonetheless.

Saving the best for last: MCODE Functions. (Set#3)

A vastly superior approach as it was mentioned before, the MCODE implementation removes all the dependencies derived from using the OS status registers to run the FOCAL program; such as PC counters, program line numbers, etc. They are also at least 20-fold faster in execution, and lastly provide a more intuitive user interface and protection safeguards against accidental data loss.

The set provided here is all based on the general-purpose code written by Monte Dalrymple. With his design it was very simple to provide canned solutions to deal with the different sections of the memory, such as the Status Registers, and the Main- or Extended memory blocks.

The function set provides choices for the particular section to back-up or restore (Main & Extended memory, Status registers), so you don't have to worry about physical memory boundaries or type of transfer. Besides those, two general-purpose functions **YMMOVE** and **YMSWAP** are also available that allow a *flexible selection of memory sectors to be transferred*, defining a sector as a 256-word set within the block – corresponding to 16 data registers, each 16-bits long.

In general, the block is the input parameter (which can be either the source or destination, depending on the function used) is prompted by the function and should be entered there in RUN mode. For Program mode execution the input parameter should be in X instead, as a decimal number 0, 1, 2, 3.

This applies to all functions except the general-purpose ones, which require both source and destination parameters to be in a control NNN stored in the X-register. Because these two functions are not prompting, this is equally so for RUN and Program modes. Note that the minimum amount of data transferred is 256 words.

The structure of the control NNN is shown below:

Source and Destination:
     input X<13> = source
     input X<12> = destination
     (Valid values are 0/1/2/3)

Data fields:
     input X<3:0>   = blocks to transfer,
     one digit (nibble) per 256-word block
     (16 sectors for complete block)

The table below shows the values used on the standard functions internally, where "b" denotes the alternate block# (1,2, or 3) used for the backup or restore:

Function	Swap Flag	Control NNN
MM-YBK	Clear	0b-00000000-00F9
MMYSWP	Set	0b-00000000-00F9
ST-YBK	Clear	0b-00000000-0001
STYSWP	Set	0b-00000000-0001
XM-YBK	Clear	0b-00000000-FF06
XMYSWP	Set	0b-00000000-FF06
YBK-MM	Clear	b0-00000000-00F9
YBK-ST	Clear	b0-00000000-0001
YBK-XM	Clear	b0-00000000-FF06
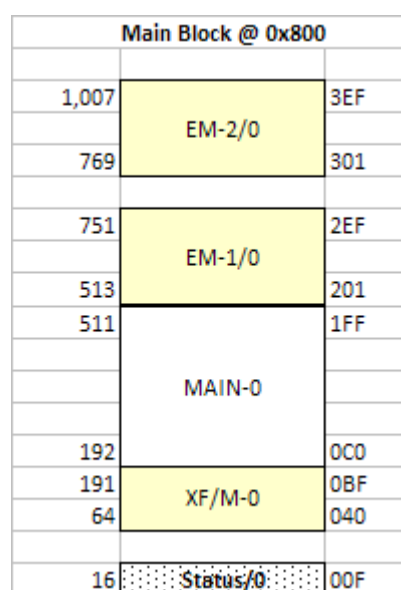
The diagram on the right shows the beginning and ending addresses for each of the significant sections within the main block; in decimal to the left and hex to the right.

With exception of the general-purpose functions **YMMOVE** and **YMSWP**, the appropriate addresses are automatically chosen by each of the memory transfer functions, completely transparent to the user.

Main Block @ 0x800

1,007	EM-2/0	3EF
769		301
751	EM-1/0	2EF
513		201
511	MAIN-0	1FF
192		0C0
191	XF/M-0	0BF
64		040
16	Status/0	00F

Lastly, to prepare the control NNN to use with the general-purpose versions of the memory transfer functions, you can use any of the HEX Entry functions available in the PWRCL_EXT, such as **CDE**, **HEXKB**, **HXNTRY**, as well as the powerhouse memory editor **RAMED**. See their description in the PWERCL_EXT Manual.

To prevent accidental data loss, the operation requires the confirmation string "OK" in ALPHA to proceed if the destination block is zero (i.e. the main RAM used by the OS). These functions are fully programmable, but you should be careful about altering the main memory (or status registers) during a program; as you may be overwriting the program itself, or other OS parameters like the program pointer or line number. This will likely result in a MEMORY LOST event.

The picture below reflects the actual addresses used by this method (and also the FOCAT sets):

Main BLOCK @ 0x800

1,007	EM-2/0	3EF
769		301
	2F0-300	
751	EM-1/0	2EF
513		201
511		1FF
	MAIN-0	
192		0C0
191	XF/M-0	0BF
64		040
	010-03F	
16	**Status/0**	00F

BLOCK-1 @ 0x801

2,031	EM-2/1	7EF
1,794		701
	6F0-730	
1,775	EM-1/1	6EF
1,537		601
1,535		5FF
	MAIN-1	
1,216		4C0
1,215	XF/M-1	4BF
1,088		440
	410-43F	
1,040	**Status/1**	40F

BLOCK-2 @ 0x802

3,055	EM-2/2	BEF
2,817		B01
	22F0-2300	
2,779	EM-1/2	AEF
2,561		A01
2,559		9FF
	MAIN-2	
2,240		8C0
2,239	XF/M-2	8BF
2,112		840
	810-83F	
2,064	**Status/2**	80F

BLOCK-3 @ 0x803

4,079	EM-2/3	FEF
3,841		F01
	EF0-F00	
3,823	EM-1/3	EEF
3,585		E01
3,583		DFF
	MAIN-3	
3,264		CC0
3,263	XF/M-3	CBF
3,136		C40
	C10-C3F	
3,087	**Status/3**	C0F

The YMEM Launcher

Granted, all these function names and purposes are both similar and inter-related, thus not surprisingly making hard to remember their scope and spelling. To palliate this, the **YMEM** launcher is included in the module, grouping the 10 MCODE functions into two launch prompts, toggled by Shift.

The two screens and the accessed functions are shown below:

Y M E M M:X:T:V:S:P:?	Y BLK M:X:T:V:S:P:?

Functions on Main block (block-0)   Functions on Expanded Memory Blocks

Hotkey	Function
M	MM-YBK _
X	XM-YBK _
T	ST-YBK _
Y	MMYSWP _
S	STYSWP _
W	XMYSWP _
P	*"BKRST"*

Hotkey	Action
[USER]	Calls **-STKSWP**
[PRGM]	Calls **YF#**
[ALPHA]	Calls **YF$**
[ENTER^]	Sub-Func. Catalog
[RADIX]	Last Function
[ON]	Calculator Off

Hotkey	Function
M	YBK-MM _
X	YBK-XM _
T	YBK-ST _
V	YMMOVE
S	YMSWAP
W	YBKSWP_
P	*"BKSWP"*

As you can see the functions on each screen are arranged to be roughly inverses of each other – within rhyme and reason of course. The Upload and Download functions (using the Serial Link) are not part of the same theme but it's useful to have them handy nevertheless.

Sub-functions and Launchers.

Pressing the keys RADIX , USER , ALPHA , and PRGM from within the **YMEM** launcher connects with the Sub-function facilities also available in this Module to access the Stack Register Swap functions, which are described next.

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, **YF# _ _ _** (prompts for index)

- ALPHA accesses the Alphanumeric Launcher, **YF$ _"** (prompts for name)

- RADIX accesses the **LASTF** feature to recall the last-executed sub-function

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 (**YF#**) or their names (**YF$**). And in addition to this method, the header function **–STKSWP** also doubles as another launcher, dedicated to this set.

In fact both launchers **YMEM** and **–STKSWP** are interconnected, so you can toggle back and forth between them by pressing the USER key:

← →

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

#	Hotkey	Sub-function		#	Hotkey	Sub-function		Hotkey	Action
1	[A]	**a<>** _ _		9	[T]	**T<>** _ _		[USER]	Toggels to YMEM
2	[B]	**b<>** _ _		10	[L]	**L<>** _ _		[PRGM]	Calls **YF#**
3	[C]	**c<>** _ _		11	[M]	**M<>** _ _		[ALPHA]	Calls **YF$**
4	[D]	**d<>** _ _		12	[N]	**N<>** _ _		[SHIFT]	**IND** *registers*
5	[E]	**e<>** _ _		13	[O]	**O<>** _ _		[RADIX]	STACK registers
6	[K]	**}-<>** _ _		14	[P]	**P<>** _ _		**[EEX]**	Adds "1_ _" to field
7	[Y]	**Y<>** _ _		15	[Q]	**Q<>** _ _		**[+]**	Upwards rotation
8	[Z]	**Z<>** _ _		-	[X]	Calls **X<>** _ _		**[-]**	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 **YF#** 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 **YF#**
02 **8** (eighth-sub-function)
03 **133** (five plus 128 for indirect)

01 X<> Z
02 X<> IND 05
03 X<> Z

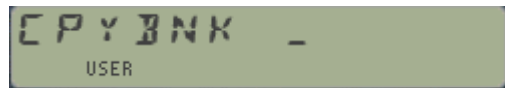## *Final Bonus: Copying code from bank-switched ROMS.*

Here's a last-minute addition to the CLXMEM module – not related to the Expanded memory subject but actually rather interesting per-se. The function code is long enough so there's no chance to include it in the PowerCL module; therefore it's added here as a second best place to include it in.

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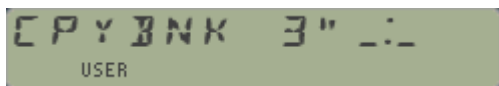
CPYBNK is a prompting function. It has a customized prompt with three distinct sections that are shown on the screen as the data entry progresses. The parameters entered are as follows:

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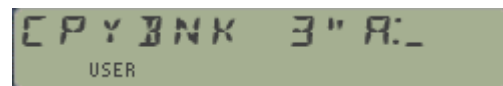
```
CPYBNK _
     USER
```

The function is smart enough to know what the first prompt must be, 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:

```
CPYBNK 3" _:_          CPYBNK 3" A:_
     USER                     USER
```

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.

```
NO BANK 4
     USER
```

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).

CL XMEM Module Manual

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.

#	Label	Sublabel	Addr	Code	Instruction	Comment
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#

#	Label	Sub	Addr	Code	Mnemonic	Comment
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#

#	Label	Label2	Addr	Code	Mnemonic	Comment
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**

What follows are the Extended Registers manipulation routines. Note how there's a CL-detection included to avoid Memory Lost issues if you use these functions on non-CL models (or emulators, like V41)..-

Header		AC6B	087	**"G"**	
Header		AC6C	012	**"R"**	***Clear Extended Registers***
Header		AC6D	018	**"X"**	
Header		AC6E	00C	**"L"**	
Header		AC6F	003	**"C"**	*Ángel Martin*
CLXRG	**AC70**		*36D*	*PORT DEP:*	*check for CL mem*
	AC71		*08C*	*XQ*	*to avoid memory lost!*
	AC72		*372*	*->A772*	*[CLMEM?]*
	AC73		130	LDI S&X	
	AC74		**3FF**	**first X-REG**	*x-curtain*
	AC75		106	A=C S&X	
	AC76		166	A=A+1 S&X	*starts at "400"*
	AC77		0A6	A<>C S&X	
	AC78		270	RAMSLCT	*select register*
	AC79		0A6	A<>C S&X	
	AC7A		04E	C=0 ALL	
	AC7B		2F0	WRTDATA	*clear it*
	AC7C		21C	PT= 2	
	AC7D		210	LD@PT- 8	*"800"*
	AC7E		306	?A<C S&X	*last one?*
	AC7F		**3BF**	**JC -09**	no, do next
	AC80		*3C1*	*?NC GO*	*yes, do a proper exit*
	AC81		*002*	*->00F0*	*[NFRPU]*

CLMEM?	**A772**	130	LDI S&X	*point at on-chip peripheral*
	A773	3F0	CON:	*3F0*
	A774	270	RAMSLCT	
	A775	3F0	PRPHSLCT	
	A776	*2A1*	*?NC XQ*	*Load RG address*
reading	A777	*12C*	*->4BA8*	*[804018]*
O setting	A778	150	LD@PT- 5	"read" command code
	A779	**1FC**	**WCMD**	*read stored turbo at 0x804018*
	A77A	*0B8*	*READ 2(Y)*	*get the value*
	A77B	2E6	?C#0 S&X	
	A77C	*381*	*?NC GO*	
	A77D	*00A*	*->02E0*	*[ERRNE]*
	A77E	*149*	*?NC GO*	*Select Chip0*
	A77F	*026*	*->0952*	*[ENCP00]*

Header	AC2A	0BE	**">"**	
Header	AC2B	03C	**"<"**	**Extra-Reg Exchange**
Header	AC2C	*118*	**"X"**	*from 1 to 5*
Header	AC2D	*118*	**"X"**	*Ángel Martin*
XX<> _ _ _	**AC2E**	**248**	**SETF 9**	
	AC2F	*063*	JNC +12d	
Header	AC30	08C	**"L"**	
Header	AC31	003	**"C"**	**Extra-Reg Recall**
Header	AC32	*112*	**"R"**	*from 1 to 5*
Header	AC33	*118*	**"X"**	*Ángel Martin*
XRCL _ _ _	**AC34**	**104**	**CLRF 8**	
	AC35	04B	JNC +09	
Header	AC36	08F	**"O"**	
Header	AC37	014	**"T"**	**Extra-Reg Storage**
Header	AC38	113	**"S"**	*from 1 to 5*
Header	AC39	118	**"X"**	*Ángel Martin*
XSTO _ _ _	**AC3A**	**244**	**CLRF 9**	
	AC3B	108	SETF 8	
	AC3C	0F8	READ 3(X)	
	AC3D	070	N=C ALL	
	AC3E	*36D*	*PORT DEP.*	*check for CL mem*
	AC3F	*08C*	*XQ*	*to avoid memory lost!*
	AC40	*372*	*->A772*	*[CLMEM?]*
	AC41	04C	?FSET 4	*SST'ing a program*
	AC42	01F	JC +03	*yes, go there*
	AC43	2CC	?FSET 13	*RUN'ing a program*
	AC44	**093**	**JNC +18d**	*no, jump over*
SST/PGM	AC45	*179*	*?NC XQ*	*Get Parameter from NextLine*
	AC46	*10C*	*->435E*	*[GETRG#]*
	AC47	130	LDI S&X	*if we were using all blocks*
	AC48	**3FF**	**max = 1,023**	*could be: C00 = (FFF - 3FF)*
	AC49	226	C=C+1 S&X	*plus one*
	AC4A	306	?A<C S&X	*ilegal index?*
	AC4B	*381*	*?NC GO*	*yes, abort program*
	AC4C	*00A*	*->02E0*	*[ERRNE]*
	AC4D	**073**	**JNC +14d**	
ILEGAL	AC4E	066	A<>B S&X	
	AC4F	130	LDI S&X	
	AC50	**200**	**offset to HEX code**	
	AC51	206	C=C+A S&X	
	AC52	1F6	C=C+C XS	
	AC53	236	C=C+1 XS	*"A5:3C/3D/3E"*
	AC54	*329*	*?NC GO*	*Complete id# and [RAK70]*
	AC55	*132*	*->4CCA*	*[RAK704]*
RUNMOD	AC56	130	LDI S&X	*if we were using all blocks*
	AC57	**3FF**	**max = 1,023**	*could be: C00 = (FFF - 3FF)*
	AC58	226	C=C+1 S&X	*plus one*
	AC59	306	?A<C S&X	*ilegal index?*
	AC5A	*3A3*	*JNC -12d*	*yes, prompt again*
MERGE0	AC5B	206	C=C+A S&X	*add index#*
	AC5C	270	RAMSLCT	*select register*
	AC5D	*038*	*READATA*	*current extra-reg*
	AC5E	10C	?FSET 8	*storage action?*
	AC5F	**027**	**JC +04**	*yes, branch off*
	AC60	0EE	C<>B ALL	
	AC61	*0B9*	*?NC GO*	*SELECTS Chip0 and Recall*
	AC62	*04A*	*->122E*	*[RCL]*
	AC63	0F0	C<>N ALL	
	AC64	*2F0*	*WRTDATA*	
	AC65	046	C=0 S&X	
	AC66	270	RAMSLCT	*select Chip0*
	AC67	24C	?FSET 9	
	AC68	3A0	?NC RTN	
	AC69	0B0	C=N ALL	
	AC6A	*3B3*	*JNC -10d*	

Appendix: Program listings. Function Set #1: Using PEEKR / POKER

1.1- Exchanging Memory Blocks (Main, X-Mem)

1	LBL "MMSWP			49	LBL 01	
2	CF 00	flag case		50	PEEKR	get source
3	?RAM			51	X<> Z(1)	
4	XEQ 10	initialize		52	PEEKR	get destination
5	"MAIN"			53	R^	put for compares
6	AVIEW			54	X=Y?	same value?
7	E	just one register (+)		55	SF 03	yes, mark for later
8	STO M(5)	size in M		56	RDN	
9	E1	register 10(+) adr		57	FS?C 03	were equal?
10	XEQ 03	swap blocks contents		58	GTO 02	yes, skip writing
11	3	3 registers: c, d, and e		59	X<> T(0)	
12	STO M(5)	block size in M		60	POKER	
13	13	register 13(c) adr		61	RDN	
14	XEQ 03	swap blocks contents		62	X<> Z(1)	
15	320	320 registers (all MM)		63	POKER	
16	STO M(5)	size in M		64	LBL 02	skip point
17	192	beginning of MM adr		65	RDN	
18	XEQ 03	swap blocks contents		66	E	
19	LBL 05	exit point		67	ST+ Z(1)	
20	"DONE"			68	+	
21	AVIEW			69	DSE M(5)	
22	RTN	all done.		70	GTO 01	next register
23	LBL "XMSWP			71	RTN	block done
24	SF 00	flag case		72	LBL 10	initialization
25	XEQ 10	initialize		73	CF 03	reset
26	"XF/M"			74	"BLK_ 123"	
27	64	beginning of block adr		75	PMTK	choose destination
28	XEQ 00			76	FC? 00	Main?
29	"EM-1"			77	XROM "OKM?"	write valid MM token
30	SF 03			78	FC? 00	Main?
31	513	beginning of block adr		79	XROM "OKM"	write valid MM token
32	XEQ 00			80	FS? 00	X-Mem?
33	"EM-2"			81	XROM "OKX?"	checks for valid XM set
34	SF 03			82	FS? 00	X-Mem?
35	769	beginning of block adr		83	XROM "OKX"	write valid XM token
36	XEQ 00			84	1024	bytes per kB
37	GTO 05	done.		85	*	block offset
38	LBL 00			86	STO 00	save for later
39	AVIEW			87	END	.
40	X<> M(5)					
41	128	block size if XF/M				
42	FS?C 03					
43	239	block size if EM-1/2				
44	X<> M(5)					
45	LBL 03	swap blocks contents				
46	RCL X(3)					
47	RCL 00	block offset				
48	+					

1.2.   Main Memory Backups & All RAM Exchange.

#	Instruction	Comment	#	Instruction	Comment
1	LBL "RAMSWP"		50	GTO 00	
2	?RAM	running from RAM?	51	LBL "MMRCL"	
3	XEQ 10		52	CF 00	recall flag
4	XROM "OKM?"	check MM token	53	?RAM	running from RAM?
5	XROM "OKX?"	check XM token	54	LBL 00	
6	XROM "OKM"	brand MM as OK	55	XEQ 10	
7	XROM "OKX"	brand XM as OK	56	FS? 00	
8	"80"		57	XROM "OKM"	brand MM as OK
9	ARCLI		58	FC? 00	
10	ASTO X		59	XROM "OKM?"	check MM token
11	"|->805"		60	"MAIN"	
12	AVIEW		61	AVIEW	
13	YMCPY		62	1024	
14	"800>"		63	•	
15	ARCL X		64	STO N	
16	AVIEW		65	192	
17	YMCPY		66	ENTER^	
18	"805>800"		67	320	
19	GTO 02		68	XEQ 03	
20	LBL "RAMSTO"		69	E1	
21	SF 00	STO flag	70	ENTER^	
22	GTO 00		71	E	
23	LBL "RAMRCL"		72	XEQ 03	
24	CF 00	RCL flag	73	13	
25	LBL 00		74	ENTER^	
26	XEQ 10		75	3	
27	FS? 00	store?	76	XEQ 03	
28	GTO 03	yes, jump over	77	LBL 02	
29	XROM "OKM?"	see if MM is OK	78	"DONE"	
30	XROM "OKX?"	check XM token	79	AVIEW	
31	GTO 04		80	RTN	
32	LBL 03	STO	81	LBL 03	
33	XROM "OKM"	brand MM as OK	82	STO M	
34	XROM "OKX"	brand XM as ok	83	X<>Y	
35	LBL 04		84	RCL N	
36	"800>80"		85	X<>Y	
37	ARCLI		86	+	
38	FC? 00	recall?	87	LASTX	
39	ASWP>	yes, swap text	88	FC? 00	
40	LBL 02		89	X<>Y	
41	AVIEW		90	LBL 01	
42	YMCPY		91	PEEKR	
43	GTO 02		92	X<>Y	
44	LBL 10		93	RDN	
45	"BLK_ 123"	prompt choices	94	POKER	
46	PMTK	request input	95	X<> T	
47	RTN		96	E	
48	LBL "MMSTO"		97	ST+ Z	
49	SF 00	store flag	98	+	
			99	DSE M	
			100	GTO 01	
			101	END	

1.3 X-Mem Backups & Memory Tokens

1	LBL "XMSTO"		52	FC? 01	
2	SF 01		53	XROM "OKX?"	
3	GTO 00		54	1024	
4	LBL "XMRCL"		55	•	
5	CF 01		56	STO 00	
6	LBL 00		57	END	
7	XEQ 10		1	LBL "OKM"	assumes bk# in X
8	"XF/M"		2	"80"	
9	64		3	ARCLI	
10	XEQ 00		4	"⌐-010-1111"	MM cntl adr
11	"EM-1"		5	YPOKE	
12	SF 00		6	RTN	
13	513		7	LBL "OKX"	assumes bk# in X
14	XEQ 00		8	"80"	
15	"EM-2"		9	ARCLI	
16	SF 00		10	"⌐-300-1111"	
17	769		11	YPOKE	
18	XEQ 00		12	RTN	done.
19	"DONE"		13	LBL "OKM?"	assumes bk# in X
20	AVIEW		14	CF 02	
21	RTN		15	"80"	
22	LBL 00		16	ARCLI	
23	AVIEW		17	"⌐-01"	
24	X<> M		18	GTO 02	
25	128		19	LBL "OKX?"	assumes bk# in X
26	FS?C 00		20	SF 02	
27	239		21	"80"	
28	X<> M		22	ARCLI	
29	RCL X		23	"⌐-30"	
30	RCL 00		24	LBL 02	
31	+		25	"⌐-0-1111"	OK branding token
31	FS? 01		26	YPEEK	read current
31	X<>Y		27	ASHF	
31	LBL 01		28	ANUM	
35	PEEKR		29	ABS	
36	X<>Y		30	1111	
37	RDN		31	X=Y?	
38	POKER		32	GTO 00	
39	X<> T		33	LBL 03	
40	E		34	FS? 02	
41	ST+ Z		35	"X"	
42	+		36	FC? 02	
43	DSE M		37	"M"	
44	GTO 01		38	"⌐-M NOT OK"	
45	RTN		39	PROMPT	
46	LBL 10		40	FC? 02	
47	CF 00		41	GTO 03	persist!
48	"BLK_ 123"		42	LBL 00	exit sign
49	PMTK		43	RCL Z	
50	FS? 01		44	END	done.
51	XROM "OKX"				

2. Function Set #2: using YMCPY

2.1. Exchanging, Storing & Recalling Memory Blocks

#				#		
1	LBL "BK_XM"			53	AVIEW	
2	3	3		54	YBUILD	copy to buffer
3	GTO 00	SF 00, SF 01		55	GTO 00	
4	LBL "BK_MM"			56	LBL 01	
5	?RAM	2		57	ASTO Z	Bk as destination adr
6	2	CF 00, SF 01		58	"|- >805"	complete page
7	GTO 00			59	AVIEW	show
8	LBL "XM-BK"	1		60	YMCPY	copy to buffer
9	1	SF 00, CF 01		61	LBL 00	
10	GTO 00			62	FS? 00	is about XM?
11	LBL "MM-BK"	0		63	GTO 10	yes, go there
12	0	CF 00, CF 01		64	"0300"	bottom of Main mem
13	LBL 00			65	YBPNT	set pointer
14	X<>F			66	CLA	
15	"BLK_ 123"			67	ARCL Y	
16	PMTK			68	"|-300-0500"	320 register
17	FC? 01	to Bk?		69	AVIEW	
18	GTO 04			70	YBUILD	build up
19	FS? 00	is about XM?		71	"0034"	reg "c" adr
20	XROM "OKX?"	check XM token		72	YBPNT	
21	FC? 00	is about MM?		73	CLA	
22	XROM "OKM?"	check MM token		74	ARCL Y	
23	GTO 03			75	"|-034-000C"	3 registers
24	LBL 04			76	AVIEW	
25	FS? 00	is about XM?		77	YBUILD	
26	XROM "OKM"	brand MM		78	"0028"	reg "+" adr
27	FC? 00	is about MM?		79	YBPNT	set pointer
28	XROM "OKX"	brand XM		80	CLA	
29	LBL 03			81	ARCL Y	
30	"800"			82	"|-028-0004"	1 register
31	ASTO X	source adr		83	GTO 05	
32	ASTO Y	destination adr		84	LBL 10	XM transfer
33	ABSP			85	"0100"	
34	ARCLI			86	YBPNT	
35	FC? 01	to Bk?		87	CLA	
36	GTO 01	yes, skip		88	ARCL Y	
37	ASTO Y	Bk as source adr		89	"|-100-0200"	
38	"805000-0000"	buffer adr and zero		90	AVIEW	
39	AVIEW			91	YBUILD	
40	YMCLR	clear buffer		92	"0804"	
41	"0034"	starting at reg "c" adr		93	YBPNT	
42	YBPNT	set pointer		94	CLA	
43	CLA			95	ARCL Y	
44	ARCL Z	destination block		96	"|-804-03BC"	
45	"|-034-0F8B"	size = 3,983 bytes		97	AVIEW	
46	AVIEW	show		98	YBUILD	
47	YBUILD	copy to buffer		99	"0C04"	
48	"0028"	reg "+" adr		100	YBPNT	
49	YBPNT	set pointer		101	CLA	
50	CLA			102	ARCL Y	
51	ARCL Z	destination block		103	"|-C04-03BC"	
52	"|-028-0004"	size = 4 bytes		104	LBL 05	

105	AVIEW	
106	YBUILD	
107	FC? 01	
108	GTO 05	
109	16	
110	WSIZE	
111	R^	puts TO: adr in X
112	RCL b	PC
113	DCD	
114	"/-**"	
115	-2	
116	AROT	
117	ASHF	
118	ASHF	
119	CLX	disable stack lift
120	H>D	
121	42	add 42 bytes
122	+	in ROM addrs INCREASE!
123	"805030-"	
124	ARCLH	
125	YPOKE	return-to addr
126	LBL 05	
127	"805>"	
128	ARCL Z	
129	AVIEW	
130	YMCPY	
131	TONE 0	return here!
132	"DONE"	
133	AVIEW	
134	END	

FLASH Backup: {YWALL, YRALL}

Two functions are available to store and recall the main memory block in/from FLASH memory. Note that this is a permanent backup that will be burned in flash memory, and thus will remain even if the calculator battery is removed or depleted.

> *Warning: Make absolutely sure that you use new batteries or freshly recharged battery packs* – flash burning is very critical and always requires battery power to be available at all times. Refer to the CL manual for additional details.

You can choose whether to do just the calculator memory block 0x800 or also include the MMU registers configuration in block 0x804. A control string in ALPHA selects that, use "OK" for the memory only or "OKALL" to also include the MMU configuration.

Function	OK	OKALL
YWALL	Writes Calculator Memory	MMU and Calculator memory
YRALL	Reads Calculator memory	MMU and Calculator memory

The backups are stored in flash blocks located at 0x1FE and 0x1FF.

> **Warning:** *Writing to flash requires the YFNX module to be configured in RAM. The utility will therefore make a copy of it into the CL RAM location at 0x805 – <u>overwriting any previous contents of this block.</u>* This may be specially damaging if you're using a RAM copy of the IMDB database, so plan accordingly first!

Serial Link transfers: { DLD48, UPL48}

Two other functions are available for Serial link transfers of complete 4k-memory blocks from the 41CL to and from the PC. The functions simply take care of initializing and setting up the serial link to use 4800 baud, and prompt for the block address in sRAM to be transferred to/from the PC.

On the PC side you need to have a program to send or receive the data using the comm port, like CLWRITER and CLREADER written by Raymond Wicker. The timing is important, as it is the sequence of operation:

- For data download, first use **DLD48**, then CLWRITER on the PC
- For data upload first run CLREADER on the PC, then **UPL48**

It works best if previous to the execution you fill in the command line switches required by CLWRITER and CLREADER on the PC side – to operate within the allowed timings.

Page-Management Functions

This section documents a few page-management functions also included in the module. Use them to enquire the status and handle the data stored in Q-RAM blocks, from the CL sRAM, the MDLD2k, or other devices with similar capability.

-BANKS & CO	Lib#4 Check & Splash	n/a
BANKED	Show Banked pages	Shows ALPHA string w/ PG numbers
BANKS? _	Get Number of banks	Prompts for Page#, returns b# in X
BFREE	Show Free pages	Shows ALPHA string w/ PG numbers
BLANK? _	Is page blank?	Prompts for Page#. YES/NO, skip if true
BUSED	Show Used pages	Shows ALPHA string w/ PG numbers
CPYBNK _ "_:_	Copy Banked Block	Thriple-prompting, does the copy
CPYPG _	Copy Page	Source in X, Destination in prompt
PGROOM _	Available Space in Page	Prompts for Page#, returns b# in X
X=PG? _	Are pages equal?	Source in X, Destination in Prompt
"XRAY"	Total Calculator ROM size	Enumerates all Pages sequentially

- **BANKED** presents a colon-separated string of numbers (in hex) corresponding to those pages with a bank-switched configuration, as defined in the ROM signature characters. The official convention is not strictly followed by the (very few) authors of the few bank-switched ROMs, but the number of banks should be marked in characters 2/3/4 of the ROM signature.

  An example with both the PowerCL and the SandMath_4x4 plugged returns the following: - Can you explain the presence of the "5"? Hurry, time's ticking out!

  ,
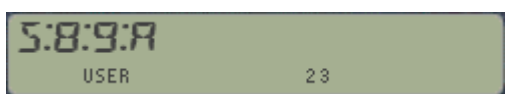
- **BFREE** and **BUSED** will present colon-separated strings of hex numbers corresponding to those free or used pages in the calculator. Obviously the OS will always be listed by **BUSED**, which is a nice clue to quickly tell which particular string you're looking at. See for instance the examples bellow showing a pretty decent configuration:

   for the free pages, and
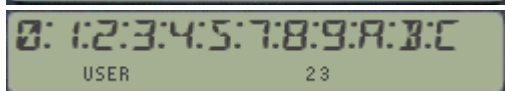
   for the used pages.

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

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

- **BLANK?** Is a test function that checks the contents of a full page, looking for non-zero words, displaying YES/NO in RUN mode accordingly. If at least one word is not zero the result is false and a program line is skipped when used in a program. Note that the word FFF is also considered to be a blank; this is used by the CL and some other MLDL boxes for "empty" Flash blocks.

- **BANKS?** returns the number of banks for a given page, which number is provided in the Hex prompt (or in X as decimal in a running program). The allowed range of results is of course 0-4: non bank-switched ROMS return a "1', and empty pages will return zero. This function reads the third nibble of the last three words in the ROM signature, which is where the bank-switched configuration is supposed to be recorded according to some undocumented criteria. This is loosely followed by the few authors who released this type of modules, thus the result may be a little off. For example, the convention used by Zengrange for the ZEPROMs is not exactly the same.

  Not even HP followed this to the letter, or if they did I cannot figure out the Advantage's scheme. Another discrepancy occurs on the CL itself, where the time signature of the TIME module has been altered – misleading **BANKS** to report 4 banks instead of just two.

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

- **X=PG?** does a block compare, sequentially checking all the bytes within the pages specified in X and the function prompt (or in X and Y if used in a program). It returns YES.NO, skipping a program line if TRUE when executed in a program.

Application Example. - How big is your lollypop?

The short program below – **XRAY** – will calculate the complete number of banks configured on-line in the calculator at a given time. I have corrected the TIME Module glitch just by starting to scan the I/O bus at page 3, thus the extra banks reported for it would account for the OS pages 0-2.

The program first enumerates the banks found for each page (set it in **TURBOX** if you want to see them), and then shows the total actual size, given in kilobytes - with 4k per page. Note that the listing shows **BANKS?** and **ARCLI** for clarity.

1	LBL "XRAY"			12	RDN
2	0			13	ISG X
3	3,015			14	GTO 00
4	LBL 00			15	X<>Y
5	"PG#"			16	4
6	ARCLI			17	*
7	"/-:"			18	CLA
8	BANKS?			19	ARCLI
9	ARCLI			20	"/-K"
10	AVIEW			21	AVIEW
11	ST+ Z			22	END