HP-41 MAXX

Every effort has been made to ensure the accuracy of the information contained herein. If you find errors or inconsistencies please bring them to our attention.

Notice:

Acknowledgements:

Ángel Martin provided the functions to use Expanded memory, the CPYBNK function and several figures in this manual. He also reviewed multiple versions of this manual and did beta testing on the module.

Sylvain Côté provided several figures used in this manual and introduced the module to the world with a presentation at HHC 2022. He also reviewed multiple versions of this manual and did beta testing on the module.

Robert Prosperi reviewed early versions of this manual and suggested including the HP-IL Code Copy functions.

Screenshots are taken from **Warren Furlow**'s V41 program.

Table of Contents

READ THIS FIRST

When First Inserting the HP-41 MAXX Module

The HP-41 MAXX module uses a Field-Programmable Gate Array (FPGA) to implement all of the logic and memory in the module. An FPGA must be loaded with the logic and memory contents at power-up, and until this process is complete the FPGA outputs are undefined. This undefined state can sometimes cause the bus drivers in the MAXX module to drive the HP-41 bus.

One particular bus signal (ISA) is used to wake up the HP-41 processor if driven while the HP-41 is off. (This is how the button on Wand works and how the Time Module displays the time continuously.) If this happens the HP-41 may wake up and then lock up when the HP-41 MAXX module is inserted into the calculator.

To clear this condition, first try pressing **ON** several times. If this doesn't work remove the batteries and then press **ON** several times before re-inserting the batteries. Insert the batteries, wait a few seconds, and then turn on the calculator. It may take a few tries for this to work, but once the calculator starts responding everything should be fine.

Neither QROM memory nor Expanded memory is initialized!

QROM blocks and Expanded memory are implemented using RAM, and are not initialized at power-up. The random data in a QROM block will almost certainly crash the calculator if the QROM block is enabled for reads prior to being initialized or loaded with actual data. Similarly, the random data in Expanded memory will likely cause problems if transferred to main memory. Here is the recommended initialization sequence to start up a calculator after first inserting the HP-41 MAXX module:

← ON	Force a Memory Clear condition, which will initialize Main memory.
QRINI	Clear all of QROM.
YMCLR 1	Clear Expanded memory block 1.
YMCLR 2	Clear Expanded memory block 2.
YMCLR 3	Clear Expanded memory block 3.
	Finally, initialize the Time module with the time and date.

All HP-41 MAXX memory is volatile!

All of the RAM in the HP-41 MAXX module is volatile and will lose its contents when power is removed.

Introduction

The HP-41 MAXX module is designed to expand an HP-41C, CV or CX calculator to the maximum capabilities envisioned for an HP-41 system. The HP-41 MAXX module implements all of these 41C modules in a single physical module:

- Four Memory modules (82106A)
- X-Functions/Memory module (82180A)
- Two X-Memory modules (82181A)
- Time module (82182A)

This module automatically scans the system every time the calculator is turned on and only activates those features not detected in the system. So for a bare-bones HP-41C all of these features will be activated, while for a HP-41CX only the two X-Memory modules will be activated. Any combination of features between these two extremes will be automatically recognized by the HP-41 MAXX module.

But the HP-41 MAXX module goes even further, adding these advanced features that were never part of the original HP-41 ecosystem:

- Three pages (each containing 1024 registers) of Expanded memory
- Twelve pages (each containing 4Kx10 bits) of RAM for instruction memory

Expanded memory (also referred to as Y-memory or Y-registers) was first introduced with the 41CL and consists of three blocks containing 1024 registers each. All three blocks can be used as a backup or alternate set of main 41C registers, with functions to easily move groups of registers between blocks. One of these blocks can be used as 1024 individual Y-registers, with a full complement of register functions.

There are twelve pages of RAM dedicated to holding code, to allow module images to be copied to RAM for editing or other purposes. Each page of this QROM (Quasi-ROM) can be assigned to any available page address (4 or 6 through 15) and to any Bank within that page. Historically, this functionality has also been referred to as MLDL memory, and this RAM can also function as HEPAX memory. Each page can be write-protected using the HEPAX write-protection mechanism.

HP-41 MAXX Configurations

The HP-41 MAXX module queries the system at each power-on and only enables those features that do not conflict with the detected system. Even though a MAXX feature may be disabled because of a change in system configuration, the contents of RAM are not lost unless power is removed.

There are nine different possibilities for how a MAXX module might be combined with other modules containing register memory in a system:

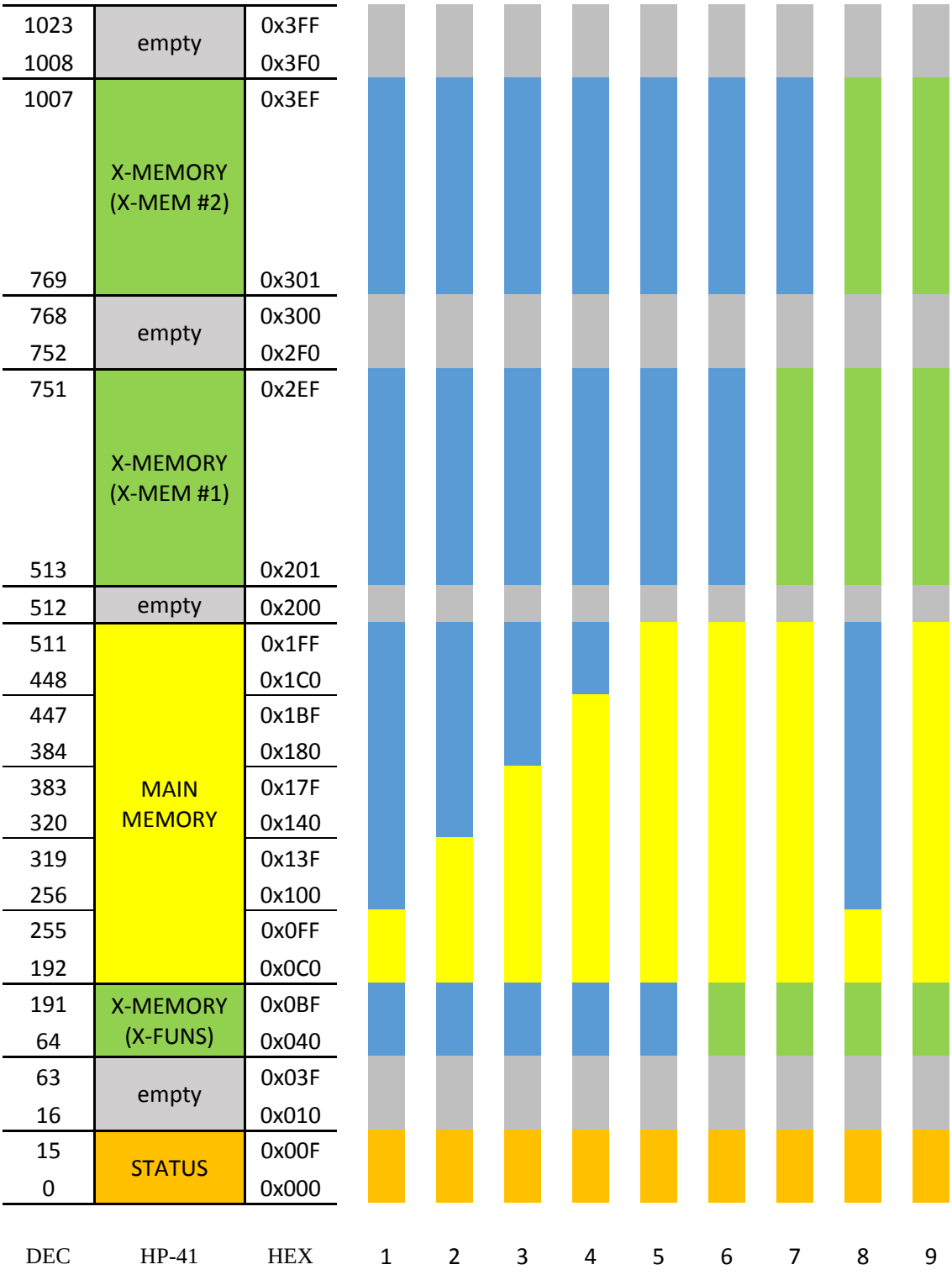
1	41C				+ MAXX
2	41C	+ MM			+ MAXX
3	41C	+ MM x2			+ MAXX
4	41C	+ MM x3			+ MAXX
5	41C	+ QM			+ MAXX
	41CV				+ MAXX
6	41C	+ QM	+ XFN		+ MAXX
	41CV		+ XFN		+ MAXX
	41CX				+ MAXX
7	41C	+ QM	+ XFN	+ XM	+ MAXX
	41CV		+ XFN	+ XM	+ MAXX
	41CX			+ XM	+ MAXX
8	41C		+ XFN	+ XM x2	+ MAXX
9	41CV		+ XFN	+ XM x2	+ MAXX
	41CX			+ XM x2	+ MAXX

(MM is an 82106A Memory Module, QM is an 82170A Quad Memory Module, XFN is and 82180A X-Functions/Memory Module, and XM is an 82181A X-Memory Module)

Because the HP-41 MAXX requires one Port, the 41C configurations are limited to three other modules. The HP-41CV includes the full compliment of regular memory, so there are fewer possible configurations with an HP-41CV. The HP-41CX lacks only the X-Memory, which limits the possible configurations even more.

Expanded memory is separate and is always enabled, which will never create a conflict because only functions in this module are capable of accessing expanded memory. This means the MAXX module is useful even in the HP-41CX case.

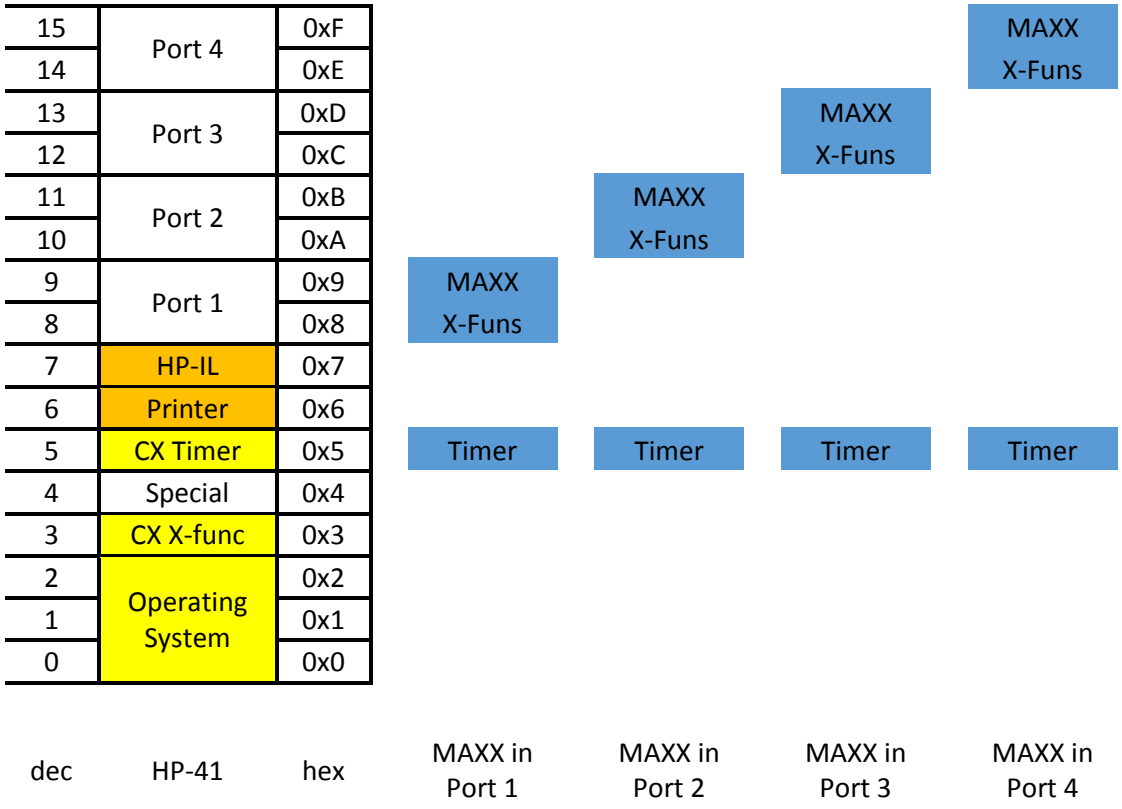
The figure below shows the HP-41 register memory organization graphically with the different possibilities for how the MAXX module may auto-configure. The MAXX contributions to the memory map are shown in blue.



Even though the MAXX module automatically compensates for other memory modules present in the HP-41, it really only makes sense to use the MAXX module by itself to keep as many ports as possible free for other uses

The HP-41 CPU employs a separate address space for instructions. This HP-41 Instruction Address space is organized into sixteen 4k pages, as shown in the left-hand column below. Three or four pages are used by the calculator itself and Page 5 is dedicated for use by the Time module. In addition, Page 6 is reserved for a printer and Page 7 is reserved for the HP-IL module. Page 4 was originally used only by the HP Service module, but recent advances allow it to instead be used to hold library code. Pairs of pages are available in each calculator port.

The columns in blue show the addresses that may be occupied by the HP-41 MAXX module. If the Timer is enabled it will occupy Page 5, while the MAXX and X-Functions software will occupy whichever Port is used to hold the module. The X-Functions code in the MAXX module cannot use Page 3 because only the 41CX Operating System software properly handles code in Page 3.

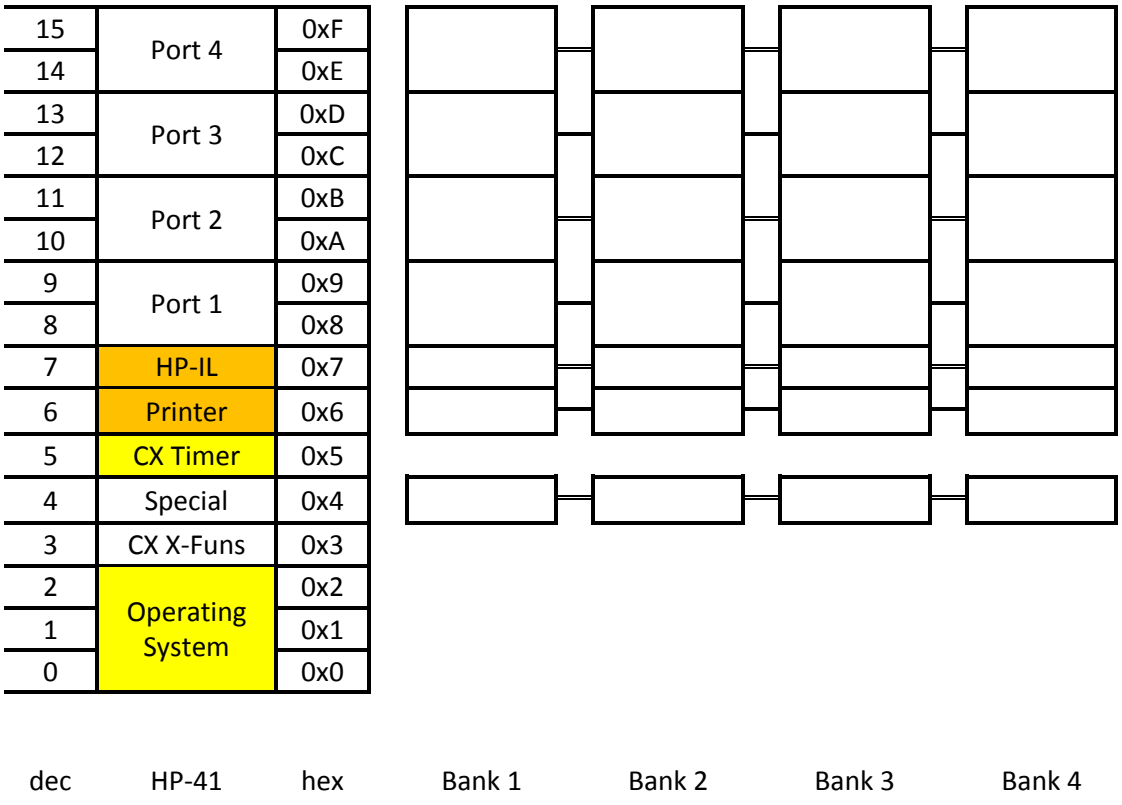


If the MAXX module is used in a 41CX the lower half of the Port where the module resides will be available for other uses, but the MAXX software always resides in the upper page of this Port.

The HP-41 Instruction memory map is actually more complicated than this, because many pages have the option of up to four banks, switched under software control. Pages 4, 6 and 7 each have their own bank-control bits, and each port also has its own bank-control bits.

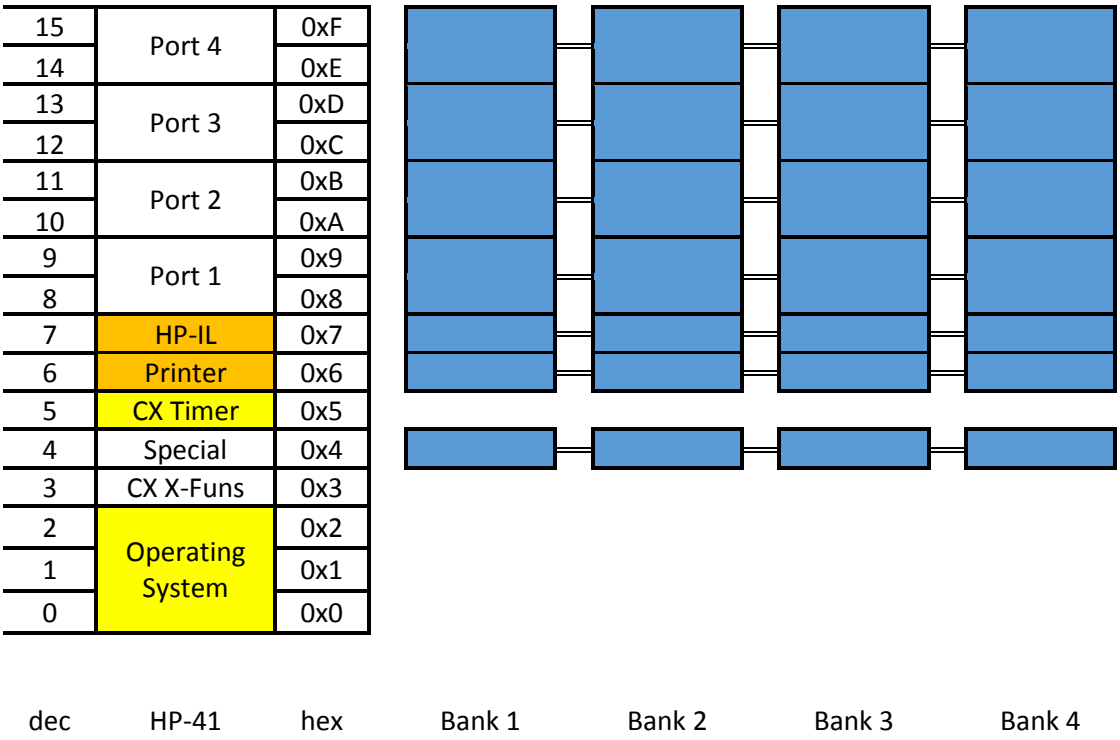
These bank-control bits are managed using the ENROM*x* mcode instructions. The four ENROM*x* instructions affect the bank-control bits that are linked to the address where the instruction is executed. The figure below shows this memory organization, with the double lines linking the various banks representing the bank-control bits.

The MAXX hardware decodes all four of the ENROM*x* instructions and implements all seven sets of bank-control bits.



Not shown in the figure above is the special bank-switching employed by the HP-41CX. In the HP-41CX pages 3 and 5 share a bank-control bit (only banks 1 and 2 are present) to allow an expanded set of X-Functions. This bank-switching is completely independent of the bank-switching in the MAXX module.

There are twelve blocks of QROM available, but each block can be programmed to occupy any available page/bank combination. This flexibility is required to cover the various uses cases, as will be shown shortly.

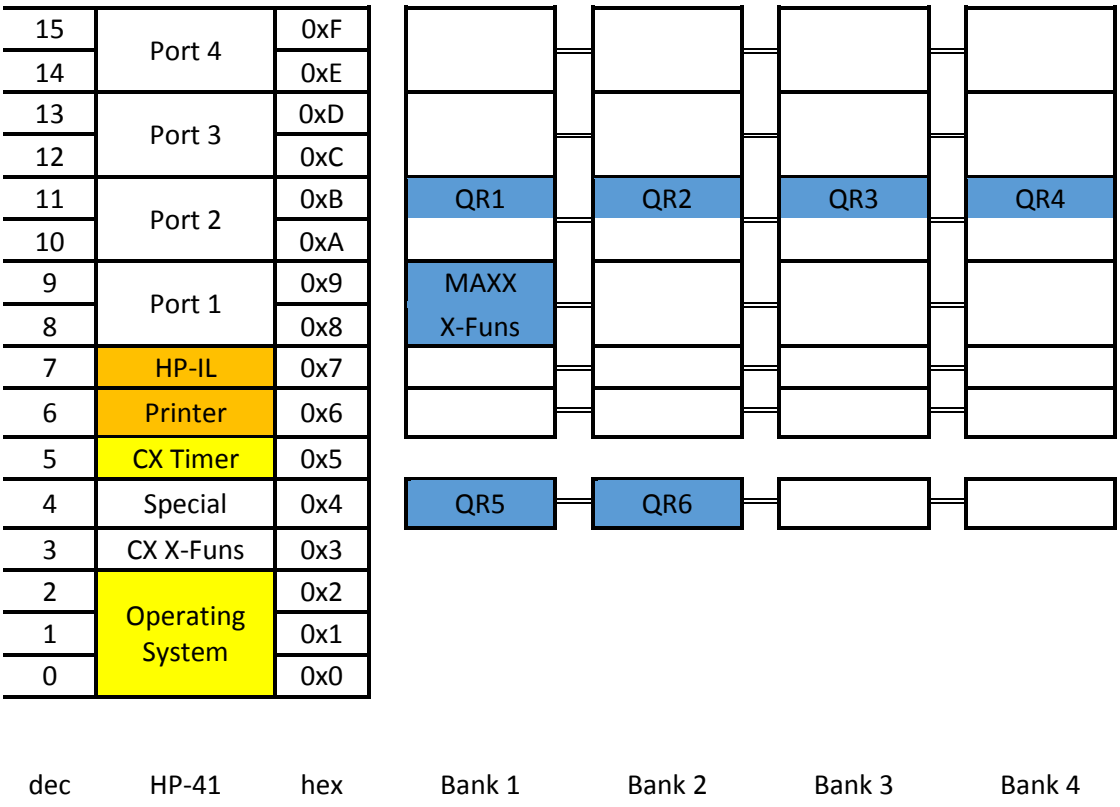


Every time the calculator is turned on software checks for collisions between physical modules and QROM, and disables any QROM blocks that would conflict with a hardware module. This collision check is only done for QROM blocks that are assigned to Bank 1. Collisions are not signaled in any way, because the user should be aware of what they plugged into the calculator.

The collision check reads the first memory location in the page, and if this location contains anything other than 0x000 the page is considered occupied. Note that when QROM is initialized (via **QRINI**) or cleared (via **QRCLR**) a 0x000 is written to all page locations, and this may confuse the collision check if QROM blocks are assigned to the same page/bank combination. This is why there is also a hardware collision check between pages of QROM. This hardware collision handling will be explained later in this section.

It is important to note that nearly all of the original software that uses Library-4, as well as Library-4 itself, assumes the presence of the 41-CX version of the Operating System. This means that these modules will only work when loaded into the HP-41 MAXX QROM if the HP-41 MAXX itself is inserted into an HP-41CX. Special versions of these modules that do not require the CX Operating System are also available.

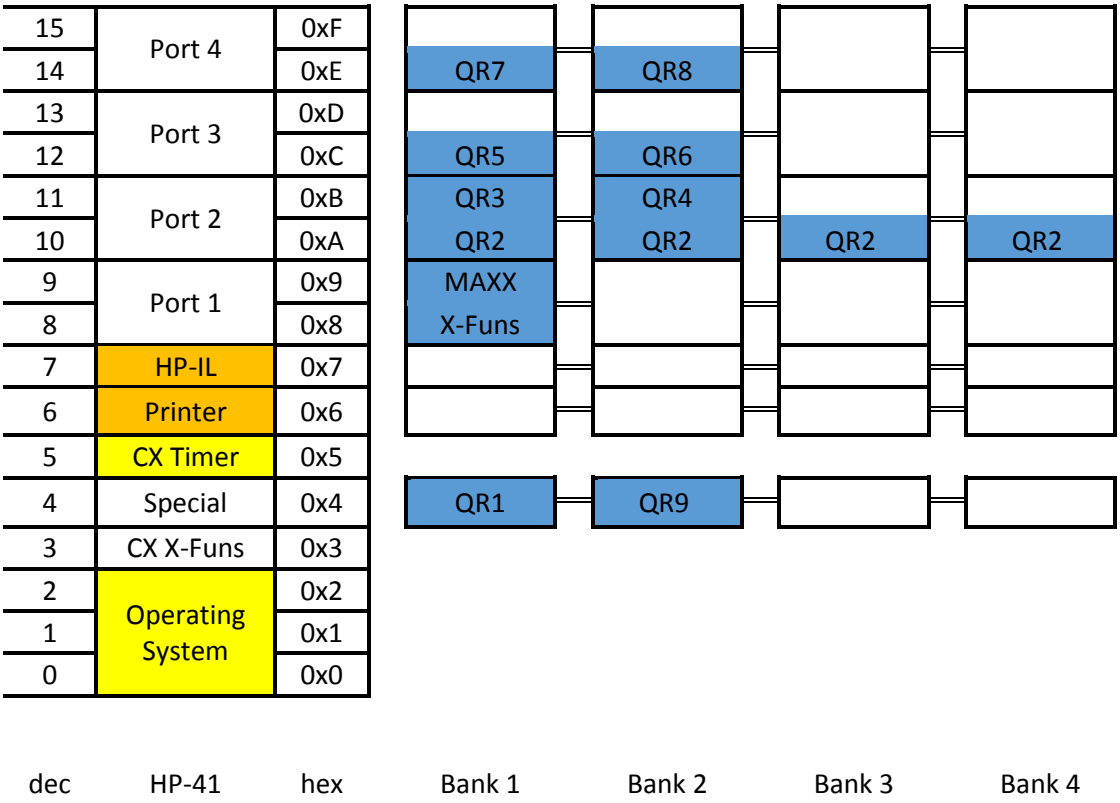
The figure below shows a simple case for the distribution of QROM blocks. In this example page 11 contains software that uses all four banks, and the fifth and sixth block of QROM are assigned to page 4, to hold library code for the 4-bank image. This still leaves six more blocks of QROM available.



The QROM programming for the figure above is as follows:

```
1 P:B:M-11:1:R      ; page 11, bank 1, read-only, active
2 P:B:M-11:2:R      ; page 11, bank 2, read-only, active
3 P:B:M-11:3:R      ; page 11, bank 3, read-only, active
4 P:B:M-11:4:R      ; page 11, bank 4, read-only, active
5 P:B:M- 4:1:R      ; page 4,  bank 1, read-only, active
6 P:B:M- 4:2:R      ; page 4,  bank 1, read-only, active
```

The figure below shows a slightly more complex case, this time with QROM blocks distributed across several different pages. Pages 11, 12, and 14 all contain code that uses two banks. The code in page 10 is not banked, so it must be programmed to ignore the state of the corresponding bank-control bits that are used by the image in page 11. The arrangement shown in Port 2 is what is used for the HP-41 Advantage Pac.



The QROM programming for the figure above is as follows:

```
1 P:B:M- 4:1:R      ; page 4,   bank 1, read-only, active
2 P:B:M-10:0:R      ; page 10, not banked, read-only, active
3 P:B:M-11:1:R      ; page 11, bank 1, read-only, active
4 P:B:M-11:2:R      ; page 11, bank 2, read-only, active
5 P:B:M-12:1:R      ; page 12, bank 1, read-only, active
6 P:B:M-12:2:R      ; page 12, bank 2, read-only, active
7 P:B:M-14:1:R      ; page 14, bank 1, read-only, active
8 P:B:M-14:2:R      ; page 14, bank 2, read-only, active
9 P:B:M- 4:2:R      ; page 4,   bank 2, read-only, active
```

This example highlights the fact that QROM blocks are completely independent, with QROM block 1 and QROM block 9 both assigned to page 4, but with different banks.

It is possible to assign QROM blocks to the same page and the same bank (as is the case after power-on) but hardware in the MAXX module automatically prioritizes accesses in these cases. This hardware priority requires some explanation, because it operates slightly differently for reads and writes.

In the case of reads, QROM blocks are prioritized from lowest number to highest number, with the lower number having the higher priority. This prioritization applies to QROM blocks assigned to the same page and bank, but only if reads are enabled. The example below shows several cases of this prioritization. In this example QROM block 2 will always be accessed when page 10 is read, independent of bank. QROM 1 would normally be higher priority, but it is not enabled for reads. QROMs 9 through 12 will never be read in this scenario because QROM 2 is programmed to be active for all banks in page 10.

```
1 P:B:M-10:0:W      ; page 10, not banked, write-only, active
2 P:B:M-10:0:R      ; page 10, not banked, read-only, active
3 P:B:M-11:1:R      ; page 11, bank 1, read-only, active
4 P:B:M-11:2:R      ; page 11, bank 2, read-only, active
5 P:B:M-12:1:R      ; page 12, bank 1, read-only, active
6 P:B:M-12:2:R      ; page 12, bank 2, read-only, active
7 P:B:M-14:1:R      ; page 14, bank 1, read-only, active
8 P:B:M-14:2:R      ; page 14, bank 2, read-only, active
9 P:B:M-10:1:B      ; page 10, bank 1, read-write, not active
10 P:B:M-10:2:B     ; page 10, bank 2, read-write, not active
11 P:B:M-10:3:B     ; page 10, bank 3, read-write, not active
12 P:B:M-10:4:B     ; page 10, bank 4, read-write, not active
```

Writes to QROM are prioritized slightly differently. The priority is still lowest number to highest number, but is broken into three sections. QROM 1-4 are in one section, QROM 5-8 are in a second section and QROM 9-12 are in the third section. The prioritization is done within each section in parallel. This was a conscious design decision, because it allows the QROM initialization function to run three times faster, by allowing three simultaneous initialization writes. This parallel write operation is only possible because each set of four QROM pages uses a separate FPGA memory instance. In the example above any write to page 10 will occur in QROM 1, with one of QROM 9-12 also being written, depending on the current bank.

MAXX Software Overview

The MAXX software is located in the upper half of the Port where the MAXX module is inserted into the HP-41 calculator. The X-Functions software (if enabled) will be located in the lower half of this Port.

The MAXX software uses XROM 15, and this XROM number cannot be changed without breaking how a number of the functions in this module work. This XROM number is used by about twenty known modules, but nearly all of them are fairly obscure.

The MAXX software is divided into eight different groups, with most groups dedicated to a specific set of hardware features.

- The MAXX Status functions report the current state of the different hardware blocks in the module, grouped so that the information will fit into the HP-41 display.

- The Expanded Memory Block functions are useful when expanded memory is being used for backups or temporary storage. These functions move blocks of data between the four blocks of expanded memory.

- The Expanded Register functions treat expanded memory Block 1 as a set of 1024 expanded memory registers, with a full complement of register-oriented functions.

- The Expanded Register Block functions are analogous to similarly-named functions available for X-memory. All of these functions operate on a block of expanded registers.

- The QROM functions control the operation of the twelve 4kx10 blocks of read/write memory that are dedicated to supporting instruction memory. QROM blocks can be assigned to page addresses between 6 and 15 or to page 4.

- The Instruction Memory functions allow direct read and write of the entire instruction address space. Only the areas of the instruction address space that support write operations, and are enabled for writes, can actually be written.

- The Code Copy function provides a way to fill QROM blocks with code from another source, typically a physical module.

- The HP-IL Code Copy functions support copying of a full page of the instruction address space to or from HP-IL, using the HP-IL Module (82160A) connected to a mass storage device. Two different algorithms for packing instruction words into bytes for HP-IL transfer are available.

MAXX Status Functions

The MAXX Status functions report the current state of the various pieces of hardware available in this module, grouped so that the information will fit into the HP-41 display. All of these functions return the status information to the display (in Run mode) as well as to the Alpha register.

MXST?

The **MXST?** (*MAXX Status?*) function returns the status of the different hardware blocks that are automatically enabled or disabled when the calculator is turned on.

The figure below shows the formatting of the status returned in the Alpha register. The eight status identifiers (**S** in the figure) will be either ***E*** (indicating that the feature is enabled) or ***D*** (indicating that the feature is disabled.) Although this status requires sixteen locations in the Alpha register the entire status fits in the display, as shown below, because the colons do not require a separate digit in the display.

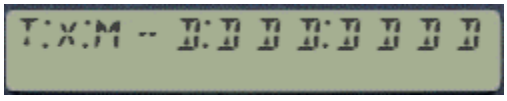
ALPHA Register															
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
T	**:**	***X***	**:**	***M***	**-**	**S**	**:**	**S**	**S**	**S**	**:**	**S**	**S**	**S**	**S**

Digit 10 reports the status for the Time module portion of the MAXX hardware.

Digits 8, 7 and 6 report the status for X-memory in 2-1-0 order. X-memory 2 corresponds to the second X-memory module in an HP-41. X-memory 1 corresponds to the first X-memory module in an HP-41. X-memory 0 corresponds to the memory in the X-Functions/Memory module.

Digits 4, 3, 2 and 1 report the status for main memory in 4-3-2-1 order. Each main memory module is numbered according to the Port where a physical module would reside.

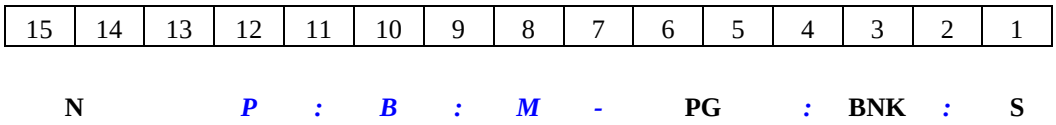
The figure below shows the formatting of the status returned in the display. This display shows that all of the MAXX hardware blocks are disabled.



QRST? (Select in X-register)

The **QRST?** (*QROM Status?*) function returns the status of the selected block of QROM. The select in the X-register can range from 0 through 12. A select value of 0 selects a CAT-like operation that displays each block status one after the other.

The figure below shows the formatting of the status returned in the Alpha register. The status identifier **N** is the number (***1*** through ***12***) of the QROM block being reported. The status identifier **PG** is a decimal number (***4*** or ***6*** through ***15***) corresponding to the page address where this block of QROM resides. The status identifier **BNK** is a number (***0*** through ***4***) that indicates the actual bank for this block of QROM. The status identifier **S** can be either ***D*** (indicating that this block of QROM is disabled), ***W*** (indicating that this block is enabled for writes only), ***R*** (indicating that this block is enabled for reads only), or ***B*** (indicating that this block is enabled for both reads and writes.)



Digits 15 and 14 show which block of QROM this report is for, which is also the input for this function. This number is right-justified in these two digit locations.

Digits 6 and 5 report the page address where this block of QROM has been assigned. This status is right-justified in these two digit locations.

Digit 3 reports which bank this block of QROM occupies. There are only five possibilities, as shown in the table below.

Digit	Bank
0	Not Banked
1	Bank 1
2	Bank 2
3	Bank 3
4	Bank 4

Digit 1 reports the read/write status for this block of QROM.

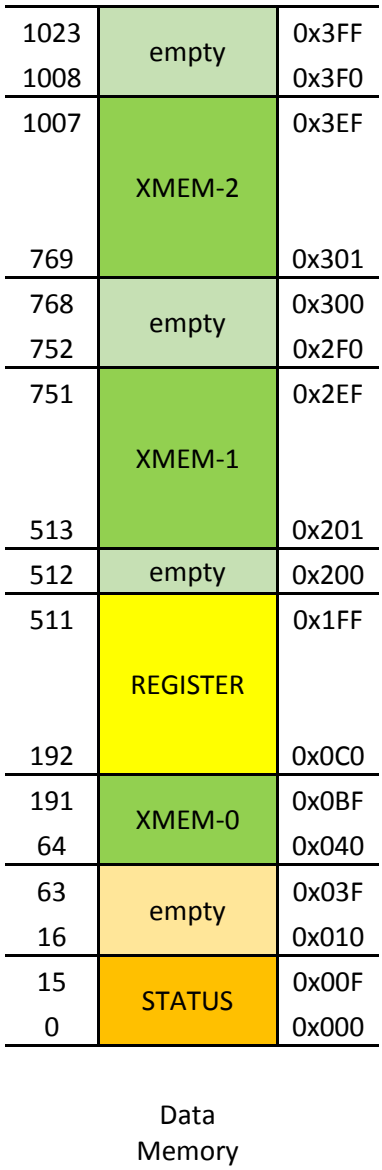
The figure below shows the formatting of the status returned in the display. This display shows the status for block 1 of QROM, which is assigned to page 15 with banking disabled, and is disabled for both reads and writes. These are the defaults (for all twelve QROM blocks) when the MAXX module is first inserted.

During the catalog operation only the **R/S** and ← keys are available any any other key will be ignored. The **R/S** key will stop or resume the listing, and a quick double-press allows single-stepping through the listing. Backward single-step is not available. The ← key terminates the catalog operation.

Expanded Memory Block Functions

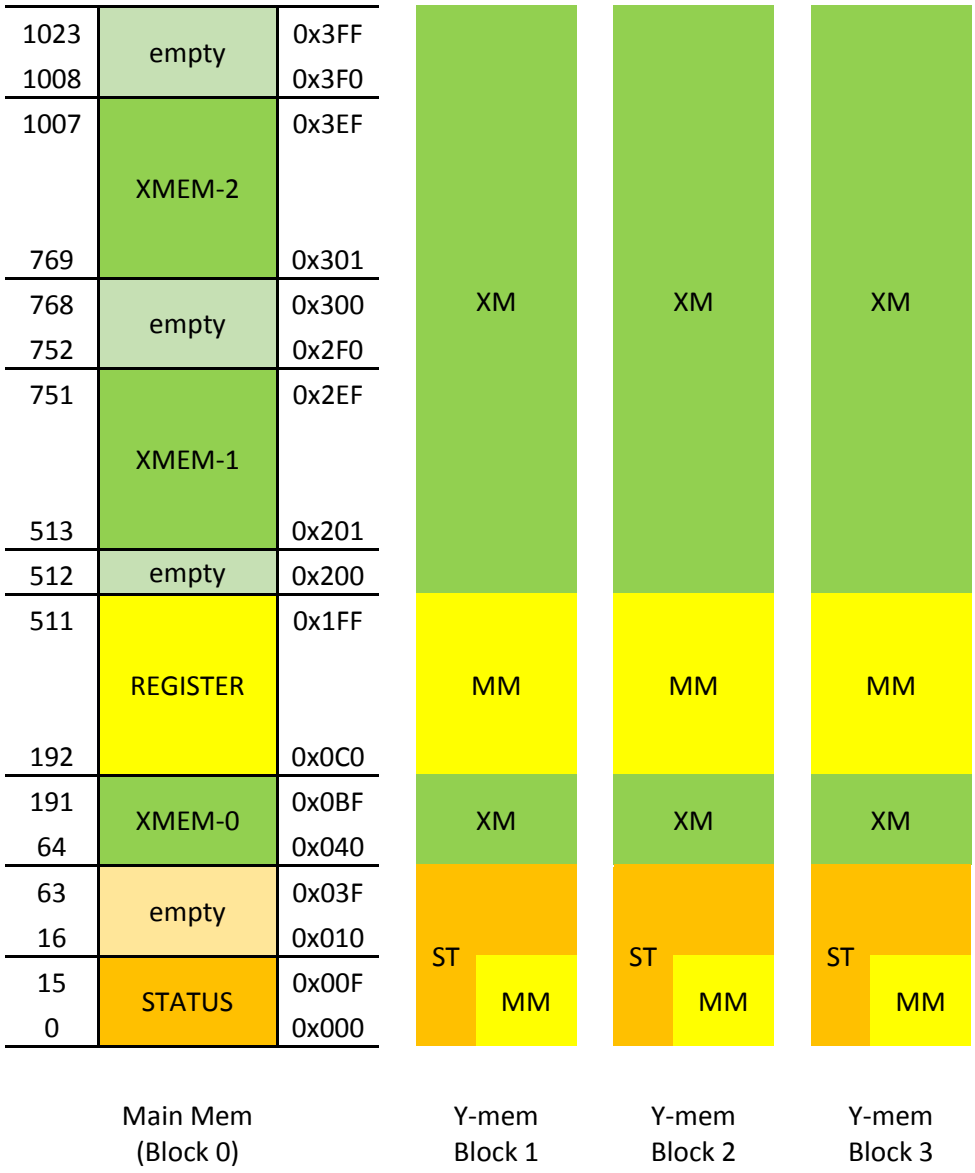
The Expanded Memory Block functions are useful when Expanded memory is being used for backups or temporary storage. The three Expanded memory blocks are numbered 1, 2 and 3, while the normal HP-41 Data memory is numbered 0 for these functions. These functions operate properly on normal HP-41 Data memory independent of whether the memory is internal or external to the HP-41 MAXX module.

The HP-41 Data memory is divided into three regions: Status (STATUS), Register memory (REGISTER) and X-memory (XMEM), as shown below.



The Status region (orange) consists of data addresses 0x000-0x03F, and contains the user-visible X, Y, Z, T and Last X registers along with the Alpha register and other registers used by the operating system. Only the lower sixteen registers are present in an HP-41, but all sixty-four registers are transferred by an ST Expanded Memory Block function.

The Register memory region (yellow) consists of the normal register and program memory of the HP-41, consisting of data addresses in the range 0x0C0-0x1FF. The MM Expanded Memory Block functions operate on this region, plus data addresses in the range 0x000-0x00F. These two blocks of register memory are the entire main memory of the HP-41.



The X-memory region (green) consists of two disjoint sections of Data memory. The section consisting of data addresses 0x040-0x0BF is the memory normally contained in an

Extended Functions/Memory module, while the section consisting of data addresses 0x200-0x3FF is the memory normally contained in a pair of X-Memory modules. The XM Expanded Memory functions operate on this area of data memory.

All of the Expanded Memory Block functions that are programmable prompt for an Expanded memory block number in Run mode and take this number from the X-register in Program mode or when Single-Stepping. Note that because of a feature in the HP-41 operating system, these functions will also prompt for a block number when entered in a program, but then discard this block number. When the program is run or single-stepped the block number is taken from the X-register.

The one non-prompting function (**YM<>YM**) takes one block number from the X-register and the other block number from the Y-register.

These functions take some amount of time to execute, as shown in the table below.

Region	Copy Execution Time	Exchange Execution Time	Clear Execution Time
ST	0.3 S	0.4 S	n/a
MM	1.5 S	2.2 S	n/a
XM	3.0 S	4.4 S	n/a
YM	4.8 S	7.0 S	1.2 S

Several of the registers in the Status area are critical for HP-41 operation, and modifying these registers can easily lead to a Memory Lost condition. To help protect against this, those Expanded Memory functions that will affect the Status area require that an ***OK*** string be present in the Alpha register before the function will execute.

The functions that require this safeguard are: **ST<>YM**, **YM>ST**, **MM<>YM**, and **YM>MM**. In addition, the **YM<>YM** function also requires this safeguard if either the source or destination memory block is Expanded Memory block 0.

Whenever a function is executed that requires this safeguard the Alpha register is automatically cleared as an additional precaution against accidentally overwriting the Status area in the future.

ST>YM (prompts for YM Block) OR (YM Block in X-register)

The **ST>YM** (*Copy Status to Expanded Memory*) function copies the Status section of the 41C Data memory (addresses 0x000-0x03F) to the corresponding locations in the selected block of Expanded memory.

ST<>YM **(prompts for YM Block) OR (YM Block in X-register)**

The **ST<>YM** (*Exchange Status with Expanded Memory*) function exchanges the Status section of the 41C Data memory (addresses 0x000-0x03F) with the corresponding locations in the selected block of Expanded memory. This function requires the string ***OK*** to be present in the Alpha register as a form of verification.

YM>ST **(prompts for YM Block) OR (YM Block in X-register)**

The **YM>ST** (*Copy Expanded Memory to Status*) function loads the Status section of the 41C Data memory (addresses 0x000-0x03F) from the corresponding locations in the selected block of Expanded memory. This function requires the string ***OK*** to be present in the Alpha register as a form of verification.

MM>YM **(prompts for YM Block) OR (YM Block in X-register)**

The **MM>YM** (*Copy Main Memory to Expanded Memory*) function copies the Main memory section of the 41C Data memory (addresses 0x0C0-0x1FF) to the corresponding locations in the selected block of Expanded memory.

MM<>YM **(prompts for YM Block) OR (YM Block in X-register)**

The **MM<>YM** (*Exchange Main Memory with Expanded Memory*) function exchanges the Main memory section of the 41C Data memory (addresses 0x0C0-0x1FF) with the corresponding locations in the selected block of Expanded memory. This function requires the string ***OK*** to be present in the Alpha register as a form of verification.

YM>MM **(prompts for YM Block) OR (YM Block in X-register)**

The **YM>MM** (*Copy Expanded Memory to Main Memory*) function loads the Main memory section of the 41C Data memory (addresses 0x0C0-0x1FF) from the corresponding locations in the selected block of Expanded memory. This function requires the string ***OK*** to be present in the Alpha register as a form of verification.

XM>YM **(prompts for YM Block) OR (YM Block in X-register)**

The **XM>YM** (*Copy Extended Memory to Expanded Memory*) function copies the X-memory section of the 41C Data memory (addresses 0x040-0x0BF and 0x200-0x3FF) to the corresponding locations in the selected block of Expanded memory.

XM<>YM **(prompts for YM Block) OR (YM Block in X-register)**

The **XM<>YM** (*Exchange Extended Memory with Expanded Memory*) function exchanges the X-memory section of the 41C Data memory (addresses 0x040-0x0BF and 0x200-0x3FF) with the corresponding locations in the selected block of Expanded memory. This function requires the string ***OK*** to be present in the Alpha register as a form of verification.

YM>XM **(prompts for YM Block) OR (YM Block in X-register)**

The **YM>XM** (*Copy Expanded Memory to Extended Memory*) function loads the X-memory section of the 41C Data memory (addresses 0x040-0x0BF and 0x200-0x3FF) from the corresponding locations in the selected block of Expanded memory. This function requires the string ***OK*** to be present in the Alpha register as a form of verification.

YM<>YM **(SRC in X-register, DST in Y-register)**

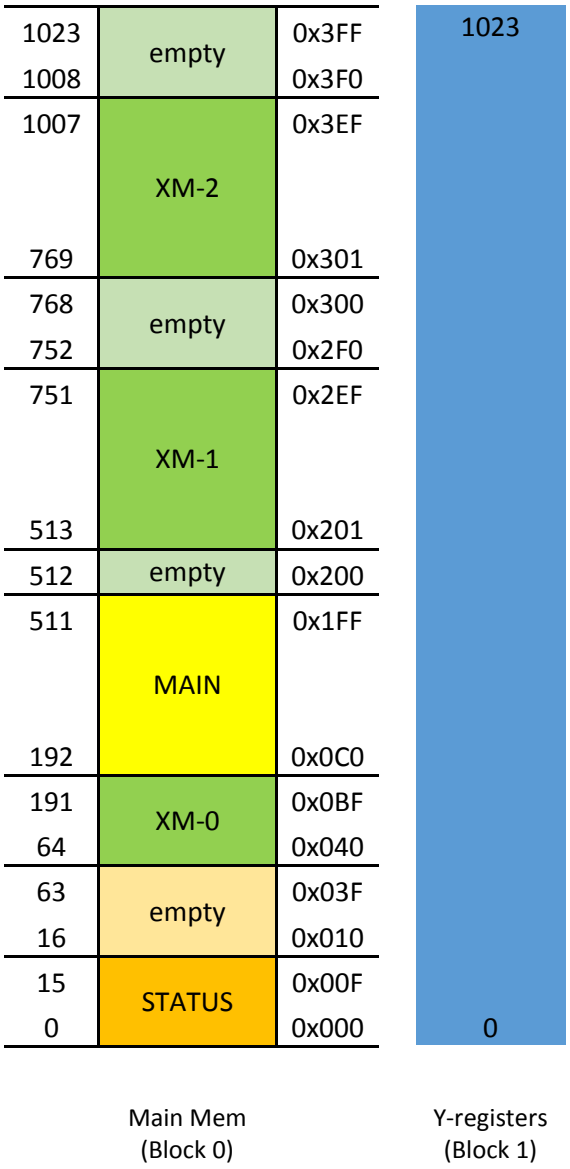
The **YM<>YM** (*Exchange Expanded Memory with Expanded Memory*) function exchanges the entire contents of two Expanded memory blocks. If either the source or the destination is Block 0, which corresponds to the normal HP-41 data memory, the function requires the string ***OK*** to be present in the Alpha register as a form of verification.

YMCLR **(prompts for YM Block) OR (YM Block in X-register)**

The **YMCLR** (*Clear Expanded Memory*) function writes zero to every location in the selected block of Expanded memory. For obvious reasons this function does not allow Block 0 (HP-41 data memory) to be cleared.

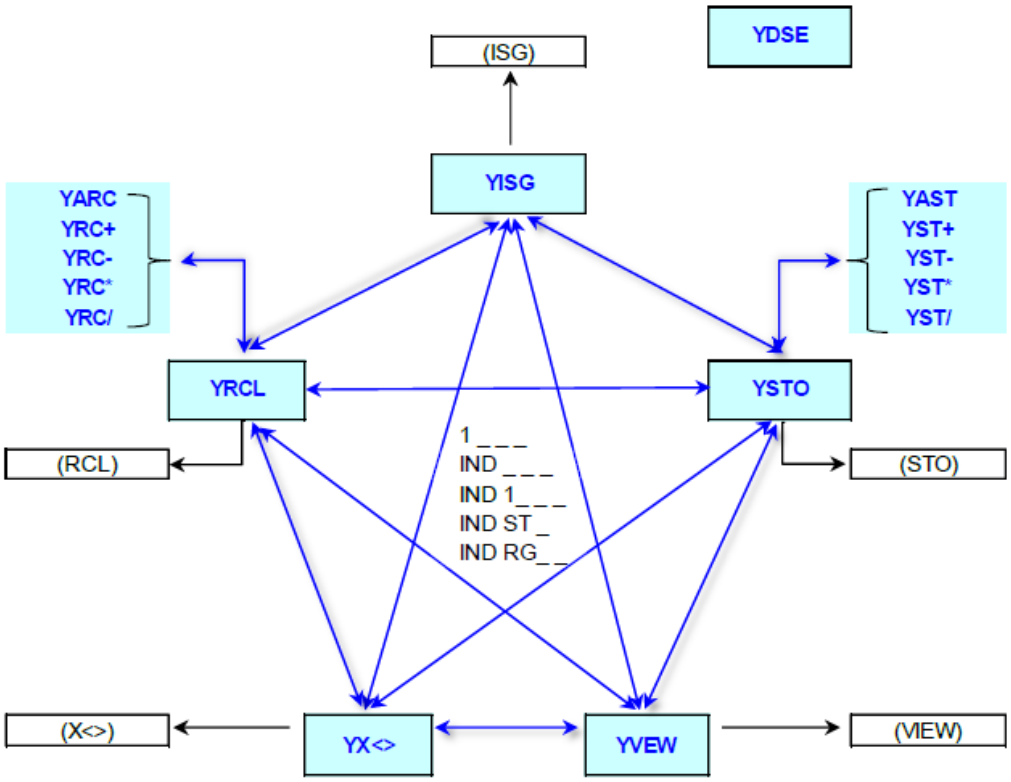
Expanded Register Functions

The Expanded Register functions treat Expanded memory Block 1 as a set of 1024 Expanded memory registers, with a full complement of register-oriented functions. These functions all prompt for a register number (000 to 1023) and also support indirect addressing, via expanded memory registers, normal registers, or the stack. All of these functions can be executed from the keyboard or entered in programs.



These function prompt for an expanded register number by displaying the function name, a quotation mark to remind the user that the function is not referring to regular registers, and three underscores as placeholders for the expanded register address. For example:

For all but one of these functions (the **YDSE** case) while these three underscores are displayed nearly the entire keyboard is available, allowing the user to move between functions, select indirect addressing or enter the expanded register address. Only the **USER**, **PRGM**, **XEQ** and **ENTER** keys are ignored in this case. The ← key terminates keyboard input and cancels the function.



The top two rows of keys provide a shortcut way to access expanded registers 001 through 010. The **Σ+** key enters 001, the **1/x** key enters 002, and so on up to the **COS** key entering 009 and the **TAN** key entering 010. By providing three digits, these keys terminate keyboard entry and start the execution of the function.

The **STO** key changes the function to **YSTO** for most cases, even if indirect addressing has been selected. However, if the current function selection is **YSTO**, then this key switches to the regular **STO** function, and this transition is irreversible.

The **RCL** key changes the function to **YRCL** for most cases, even if indirect addressing has been selected. However, if the current function selection is **YRCL**, then this key switches to the regular **RCL** function, and this transition is irreversible.

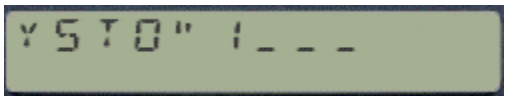
The **SST** key changes the function to **YX<>** for most cases, even if indirect addressing has been selected. However, if the current function selection is **YX<>**, then this key switches to the regular **X<>** function, and this transition is irreversible. The **SST** key is used because the normal **X<>Y** key is in the second keyboard row and is dedicated to the numeric shortcut previously described.

The **CHS** key changes the function to **YISG** for most cases, even if indirect addressing has been selected. However, if the current function selection is **YISG** or **YDSE**, then this key switches to the regular **ISG** function, and this transition is irreversible. The fact that the normal **DSE** function is not available on the keyboard is why the **YDSE** function is treated slightly differently from all of the other expanded register functions as far as the keyboard operation.

The **R/S** key changes the function to **YVEW** for most cases, even if indirect addressing has been selected. However, if the current function selection is **YVEW**, then this key switches to the regular **VIEW** function, and this transition is irreversible.

The **ALPHA** key is active for the various store and recall functions (but not **YAST** or **YARC**) and is ignored in all other cases. Any store function will be changed to **YAST** and any recall function will be changed to **YARC**, even if indirect addressing has been selected.

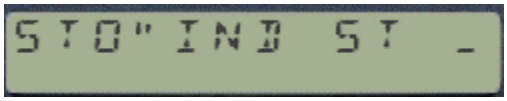
The **EEX** key is used to create a four-digit expanded register address by prepending a **1** to the address field, as shown below. Once digit entry has begun only the numeric keys (and the ← key) are active. There are only 1024 expanded registers, so the next digit must be a 0, but this is not checked until all four digits have been entered.



The arithmetic keys (**+**, **-**, ***** and **÷**) will change the operation to add the selected arithmetic operation, for either store or recall. But the arithmetic keys are ignored for the **YISG**, **YDSE**, **YVEW** and **YX<>** functions. In addition, these keys will cancel the indirect addressing mode at the same time that the function is changed.

The gold **SHIFT** key changes the addressing for the function to indirect using an expanded register, as shown below. In this case the full keyboard continues to be available, and works as described here.

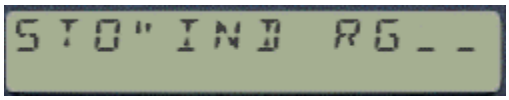
The radix key (**.**) can be used to modify the indirect addressing to use a stack register for the indirect address, as shown below.



Note that the display does not have enough room to display the entire function name in this case. The stack register is specified according to the Alpha key table below. Specifying anything other than a normal stack register (X, Y, Z, T) is not recommended for the casual user.

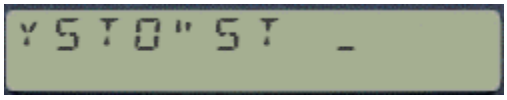
Key	Stack register selection	Address
A	Synthetic "a" register	0x00F
B	Synthetic "b" register	0x00E
C	Synthetic "c" register	0x00D
D	Synthetic "d" register	0x00C
E	Synthetic "e" register	0x00B
K	Synthetic "k" register	0x00A
L	LASTX register	0x004
M	Synthetic "M" register	0x005
N	Synthetic "N" register	0x006
O	Synthetic "O" register	0x007
P	Synthetic "P" register	0x008
Q	Synthetic "Q" register	0x009
T	T stack register	0x000
X	X stack register	0x003
Y	Y stack register	0x002
Z	Z stack register	0x001

When Indirect addressing has been selected, pressing the radix key again changes the mode to Indirect Register, as shown below. In this case a regular register address (0 to 99) is used for the indirect address.

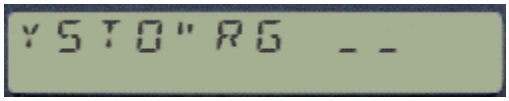


In either of these two cases the radix key will toggle between the two possibilities, while the ← key will return to the basic **YSTO** or **YRCL** function.

The radix key is active even when the **IND** indicator is not present, and allows operations on the entire range of stack registers.



As before, pressing the radix key again will toggle the function to select a regular 41C register address, while the ← key will return to the basic **YSTO** or **YRCL** function.



The direct Stack and Data Register options are useful in Run mode but it doesn't make sense to use them in programs because the equivalent native 41C function requires less program space.

When entered in a Program these Expanded Register function cases will automatically be converted to the native 41C functions, as highlighted in the table below.

Direct	Indirect	Data Register	Indirect Data Reg	Stack	Indirect Stack
YSTO nnn	YSTO IND nnn	STO nn	YSTO IND RG nn	STO ST (nn)	YSTO IND ST nn
YST+ nnn	YST+ IND nnn	ST+ nn	YST+ IND RG nn	ST+ ST (nn)	YST+ IND ST nn
YST- nnn	YST- IND nnn	ST- nn	YST- IND RG nn	ST- ST (nn)	YST- IND ST nn
YST* nnn	YST* IND nnn	ST* nn	YST* IND RG nn	ST* ST (nn)	YST* IND ST nn
YST/ nnn	YST/ IND nnn	ST/ nn	YST/ IND RG nn	ST/ ST (nn)	YST/ IND ST nn
YRCL nnn	YRCL IND nnn	RCL nn	YRCL IND RG nn	RCL ST (nn)	YRCL IND ST nn
YRC+ nnn	YRC+ IND nnn	YRC+ RG nn	YRC+ IND RG nn	YRC+ ST nn	YRC+ IND ST nn
YRC- nnn	YRC- IND nnn	YRC- RG nn	YRC- IND RG nn	YRC- ST nn	YRC- IND ST nn
YRC* nnn	YRC* IND nnn	YRC* RG nn	YRC* IND RG nn	YRC* ST nn	YRC* IND ST nn
YRC/ nnn	YRC/ IND nnn	YRC/ RG nn	YRC/ IND RG nn	YRC/ ST nn	YRC/ IND ST nn
YDSE nnn	YDSE IND nnn	DSE nn	YDSE IND RG nn	DSE ST (nn)	YDSE IND ST nn
YISG nnn	YISG IND nnn	ISG nn	YISG IND RG nn	ISG ST (nn)	YISG IND ST nn
YX<> nnn	YX<> IND nnn	X<> nn	YX<> IND RG nn	X<> ST (nn)	YX<> IND ST nn
YVEW nnn	YVEW IND nnn	VIEW nn	YVEW IND RG nn	VIEW ST (nn)	YVEW IND ST nn
YARC nnn	YRCL IND nnn	ARCL nn	YRCL IND RG nn	ARCL ST (nn)	YRCL IND ST nn
YAST nnn	YSTO IND nnn	ASTO nn	YSTO IND RG nn	ASTO ST (nn)	YSTO IND ST nn

In program mode the Expanded Register functions use two program lines, one for the function code and one for the Y-register number. In the case of Indirect, Register, Indirect Register, Stack and Indirect Stack, the register number is modified to indicate the special type of addressing. These modifications are shown in the table below. The modifications are managed automatically and these two program lines should not be modified by the user.

Function	Direct	Indirect	Data Register	Indirect Data Reg	Stack	Indirect Stack
YSTO	nnn	nnn + 1024	STO nn	nn + 2048	STO ST (nn)	nn + 3072
YST+	nnn	nnn + 1024	ST+ nn	nn + 2048	ST+ ST (nn)	nn + 3072
YST-	nnn	nnn + 1024	ST- nn	nn + 2048	ST- ST (nn)	nn + 3072
YST*	nnn	nnn + 1024	ST* nn	nn + 2048	ST* ST (nn)	nn + 3072
YST/	nnn	nnn + 1024	ST/ nn	nn + 2048	ST/ ST (nn)	nn + 3072
YRCL	nnn	nnn + 1024	RCL nn	nn + 2048	RCL ST (nn)	nn + 3072
YRC+	nnn	nnn + 1024	nn + 2560	nn + 2048	YRC+ ST nn	nn + 3072
YRC-	nnn	nnn + 1024	nn + 2560	nn + 2048	YRC- ST nn	nn + 3072
YRC*	nnn	nnn + 1024	nn + 2560	nn + 2048	YRC* ST nn	nn + 3072
YRC/	nnn	nnn + 1024	nn + 2560	nn + 2048	YRC/ ST nn	nn + 3072
YDSE	nnn	nnn + 1024	DSE nn	nn + 2048	DSE ST (nn)	nn + 3072
YISG	nnn	nnn + 1024	ISG nn	nn + 2048	ISG ST (nn)	nn + 3072
YX<>	nnn	nnn + 1024	X<> nn	nn + 2048	X<> ST (nn)	nn + 3072
YVEW	nnn	nnn + 1024	VIEW nn	nn + 2048	VIEW ST (nn)	nn + 3072
YARC	nnn	nnn + 1024	ARCL nn	nn + 2048	ARCL ST (nn)	nn + 3072
YAST	nnn	nnn + 1024	ASTO nn	nn + 2048	ASTO ST (nn)	nn + 3072

The remainder of this section describes the details of each individual Expanded Register function.

YARC (prompts for YM register)

The **YARC** (*Expanded Register Alpha Recall*) function appends the contents of the selected expanded register to the Alpha register.

YAST (prompts for YM register)

The **YAST** (*Expanded Register Alpha Store*) function copies the first six characters in the Alpha register to the selected expanded register. The Alpha register is unaffected.

YDSE (prompts for YM register)

The **YDSE** (*Expanded Register Decrement and Skip if Equal*) function operates identically to the normal HP-41C **DSE** function, except that it uses an expanded register.

YISG (prompts for YM register)

The **YISG** (*Expanded Register Increment and Skip if Greater*) function operates identically to the normal HP-41C **ISG** function, except that it uses an expanded register.

YRCL (prompts for YM register)

The **YRCL** (*Expanded Register Recall*) function copies the contents of the selected expanded register to the X-register. The stack is lifted and the expanded register is not affected.

YRC+ (prompts for YM register)

The **YRC+** (*Expanded Register Recall and Add*) function adds the contents of the selected expanded register to the contents of the X-register and places the result in the X-register. The stack is not lifted and the expanded register is unaffected.

YRC- (prompts for YM register)

The **YRC-** (*Expanded Register Recall and Subtract*) function subtracts the contents of the selected expanded register from the contents of the X-register and places the result in the X-register. The stack is not lifted and the expanded register is unaffected.

YRC* (prompts for YM register)

The **YRC*** (*Expanded Register Recall and Multiply*) function multiplies the contents of the selected expanded register with the contents of the X-register and places the result in the X-register. The stack is not lifted and the expanded register is unaffected.

YRC/ (prompts for YM register)

The **YRC/** (*Expanded Register Recall and Divide*) function divides the contents of the X-register by the contents of the expanded register and places the result in the X-register. The stack is not lifted and the expanded register is unaffected.

YSTO (prompts for YM register)

The **YSTO** (*Expanded Register Store*) function copies the contents of the X-register to the selected expanded register. The X-register is not affected.

YST+ (prompts for YM register)

The **YST+** (*Expanded Register Add and Store*) function adds the contents of the X-register to the contents of the selected expanded register and stores the result in the expanded register. The X-register is unaffected.

YST- (prompts for YM register)

The **YST-** (*Expanded Register Subtract and Store*) function subtracts the contents of the X-register from the contents of the selected expanded register and places the result in the expanded register. The X-register is unaffected.

YST* (prompts for YM register)

The **YST*** (*Expanded Register Multiply and Store*) function multiplies the contents of the selected expanded register by the contents of the X-register and places the result in the expanded register. The X-register is unaffected.

YST/ (prompts for YM register)

The **YST/** (*Expanded Register Divide and Store*) function divides the contents of the selected expanded register by the contents of the X-register and places the result in the expanded register. The X-register is unaffected.

YVEW (prompts for YM register)

The **YVEW** (*View Expanded Register*) function recalls the contents of the selected expanded register to the display, just like the normal **VIEW** function.

YX<> (prompts for YM register)

The **YX<>** (*Expanded Register Exchange with X*) function exchanges the contents of the selected expanded register with the contents of the X-register.

Expanded Register Block Functions

The Expanded Register Block functions are analogous to similarly-named functions available for X-memory. All of these functions operate on a block of expanded registers.

YRGMOV (control word in X-register)

The **YRGMOV** (*Expanded Register Block Move*) function moves the contents of one block of expanded registers to another block of expanded registers as specified by the *bbb.eeennn* control word in the X-register.

The *bbb* digits are the base address of the source expanded register block, in the range R_0 through R_{999}. The *eee* digits are the base address destination expanded register block, again in the range R_0 through R_{999}. The *nnn* digits are the number of registers to be copied. If *nnn* is zero a value of one used.

These two blocks can be overlapping, but only if *bbb* is greater than *eee*. This is because the transfers start at the highest address in the block and work downwards.

YRGSWP (control word in X-register)

The **YRGSWP** (*Expanded Register Block Swap*) function exchanges the contents of one block of expanded registers with the contents of another block of expanded registers, as specified by the *bbb.eeennn* control word in the X-register.

The *bbb* digits are the base address of the first block of expanded registers, in the range R_0 through R_{999}. The *eee* digits are the base address of the second block of expanded registers, again in the range R_0 through R_{999}. *nnn* is the number of registers to be exchanged. If *nnn* is zero a value of one used.

These blocks can be overlapping, but only if *bbb* is greater than *eee*. This is because the exchanges start at the highest address in the block and work downwards.

CLYRG

The **CLYRG** (*Clear Expanded Register Block*) function writes zero to the entire set of expanded registers, R_0 through R_{1023}.

CLYRGX (control word in X-register)

The **CLYRGX** (*Clear Expanded Register Block by X*) function writes zeros to all of the expanded registers in the range R_{bbb} to R_{eee}, inclusive. The *bbb.eee* control word is taken from the X-register.

A<>YRG (prompts for base address) OR (base address in X-register)

The **A<>YRG** (*Exchange Alpha Register with Expanded Registers*) function exchanges the contents of the Alpha register (plus the Temporary Alpha scratch register) with five expanded registers starting at the base address.

In Run mode the function prompts for a three-digit expanded register address, with a leading **1** is entered via the **EEX** key. In Program mode or Single-stepping the base address is taken from the X-register. The base register can range from R_{000} through R_{1019}.

The table below shows the arrangement of the exchanged registers.

Y-register	Exchanged register	
	A<>YRG	ST<>YRG
R_{nnn+4}	Alpha register 7-1	T stack register
R_{nnn+3}	Alpha register 14-8	Z stack register
R_{nnn+2}	Alpha register 21-15	Y stack register
R_{nnn+1}	Alpha register 28-22	X stack register
R_{nnn}	Temporary Alpha scratch	L stack register

ST<>YRG (prompts for base address) OR (base address in X-register)

The **ST<>YRG** (*Exchange Stack Registers with Expanded Registers*) function exchanges the contents of the stack registers with five expanded registers starting at the base address.

In Run mode the function prompts for a three-digit expanded register address, with a leading **1** is entered via the **EEX** key. In Program mode or Single-stepping the base address is taken from the X-register. The base register can range from R_{000} through R_{1019}.

The table under the **A<>YRG** function description shows the arrangement of the exchanged registers.

QROM Functions

QROM consists of twelve 4kx10 blocks of read/write instruction memory. Each of these twelve blocks can be assigned to a separate page address/bank combination. QROM can be assigned to any page address between 6 and 15 or to page 4. Each QROM block can be assigned to Bank 1, 2, 3 or 4, or banked operation can be disabled for the block. Each QROM block can be enabled or disabled for reads and writes separately.

When the MAXX module is first inserted into the calculator all twelve QROM blocks are assigned to Page 15, with banked operation disabled. There is no conflict with this assignment, because all twelve QROM blocks are also disabled for both reads and writes at the same time.

QROM is normally under control of the user, but every time the calculator is turned on the physical Ports are checked for conflicts with the QROM page addresses. If a module is present coincident with a QROM page address, and the QROM block is enabled for bank 1 or banked operation is disabled, the QROM block will be automatically disabled for reads. QROM blocks are never automatically enabled for reads once the conflict is removed, so the user will need to do this.

The Port conflict check is done using the MAXX software polling point. Because the HP-41 checks for polling point code starting with Page 5 and working upward, this means that anything plugged into the HP-41 at a page address lower than that of the MAXX software will be accessed prior to the conflict check. As a result, users must disable any QROM pages with a conflicting page address lower than the MAXX module prior to plugging in a physical module. This applies mostly to the Printer and HP-IL modules, because they contain polling point code that will definitely conflict with a co-located QROM image.

When first enabling QROM pages for reads, it is important to enable the blocks in the correct order. For Banked images this means that Bank 1 should be enabled last, so that the other banks are present and ready for access when the main bank is enabled. For multi-page images this means that the secondary pages should be enabled first and the primary page enabled last, for the same reason.

A QROM block that is being relocated from one page address to another page address should be disabled for reads before being moved. The page address modification operation is atomic, meaning that will be completed before there is a chance that the new addressing will be noticed by the calculator, but it is always better to be safe than sorry.

QROM blocks are implemented using RAM, and are not initialized at power-up. The random data in a QROM block will almost certainly crash the calculator if the QROM block is enabled for reads prior to being initialized or loaded with actual data.

The table below shows how the various functions affect the QROM control fields. Most of the QROM functions take the block number, in the range 1-12, from the X-register to select the QROM block to operate on. The X-register is not affected. This selection should not be confused with the HP-41 page/bank address that the QROM block is assigned to.

Global operation can be selected with a value of "0" in the X-register. Global operation applies the operation to all twelve QROM blocks at once.

Function	Address	Bank	Write Enable	Read Enable	Global operation
QRABY	Set	Set	-	-	no
QRCLR	-	-	-	-	no
QRINI	-	-	-	-	N/A
QRRE	-	-	-	Set	no
QRRO	-	-	Cleared	Set	no
QRRP	-	-	-	Cleared	yes
QRRWE	-	-	Set	Set	no
QRRWD	-	-	Cleared	Cleared	yes
QRWE	-	-	Set	-	yes
QRWO	-	-	Set	Cleared	yes
QRWP	-	-	Cleared	-	yes

QRABY (Select in X-register; control word in Y-register)

The **QRABY** (*QROM Set Address/Bank by Y-register*) function programs the page/bank address for the selected block of QROM, using the *pp.b* control word in the Y stack register. Both the X-register and Y-register are unaffected.

The *pp* digits (or digit) are the page address for the selected block. QROM can be assigned to page address 4 or any page 6 through 15. The *b* digit is the bank to be occupied by the selected block and ranges from 0 to 4. The 0 case tells the control logic to ignore the bank select bits for this block of QROM.

This function only programs the page address and bank, so read or write enables must be programmed separately. This function does check whether or not the page address is already occupied, but only if unbanked or Bank 1 is selected. An error message will be generated in the case of an occupied page.

A QROM block that is being relocated from one page to another should be disabled for reads before being moved.

QRCLR (Select in X-register)

The **QRCLR** (*QROM Clear*) function initializes the selected block (1 - 12) of QROM to all zeros, providing finer granularity than the **QRINI** function.

QRINI

The **QRINI** (*QROM Initialize*) function fills all twelve blocks of QROM with all zeros. This function should be executed when the HP-41 MAXX module is first inserted into the calculator because the QROM blocks power up containing random data.

This function requires about seven seconds to complete and the display is blank during execution.

QRRE (Select in X-register)

The **QRRE** (*QROM Read Enable*) function enables reads for the selected block of QROM, which has the effect of inserting this block into the HP-41 address space. This function returns an error message if the page is already occupied. The write enable for the QROM block is not affected.

A QROM block must always be initialized in some way before reads of the block are enabled to prevent locking up the machine. The page address and bank used by the QROM block should also be initialized prior to enabling reads.

QRRO (Select in X-register)

The **QRRO** (*QROM Read-Only*) function enables reads and disables writes for the selected block of QROM. With this combination the QROM acts like a normal read-only memory. This function returns an error message if the page is already occupied.

QRRP (Select in X-register)

The **QRRP** (*QROM Read Protect*) function disables reads for the selected block of QROM, which has the effect of removing the memory block from the HP-41 address space

as far as instruction execution is concerned. The write enable for the memory block is unaffected, and the QROM block retains its contents. Reads are disabled by default.

QRRWE (Select in X-register)

The **QRRWE** (*QROM Read and Write Enable*) function enables both reads and writes for the selected block of QROM. With this combination the QROM acts like a normal random-access memory. This mode should be used when QROM blocks are being used as HEPAX memory. This function returns an error message if the page is already occupied.

QRRWD (Select in X-register)

The **QRRWD** (*QROM Read and Write Disable*) function disables both reads and writes for the selected block of QROM, which has the effect of completely removing the memory block from the HP-41 address space. The QROM block retains its contents.

QRWE (Select in X-register)

The **QRWE** (*QROM Write Enable*) function enables writes for the selected block of QROM. Since the QROM is part of the HP-41 instruction address space, writes are normally rare. The read enable for the selected block of QROM is not affected.

QRWO (Select in X-register)

The **QRWO** (*QROM Write-Only*) function enables writes and disables reads for the selected block of QROM. This combination is useful for when a page is being loaded with an image from HP-IL.

QRWP (Select in X-register)

The **QRWP** (*QROM Write Protect*) function disables writes for the selected block of QROM. Since QROM is part of the HP-41 instruction address space, writes would normally be disabled. The read enable for the selected block of QROM is not affected.

Instruction Memory Functions

The Instruction Memory functions support direct read and write of the entire instruction address space. Only the areas of the instruction address space that support write operations, and are enabled for writes, will actually be written, but the entire instruction address space can be read.

Two different methods for specifying the address and data are supported with these functions: hexadecimal in the ALPHA register and decimal in the X-register.

To use hexadecimal address and data the address and data must be present in the ALPHA register in a specific format. The figure below shows the formatting of the address and data fields in the ALPHA register. All seven hexadecimal digits must be present.

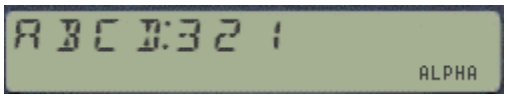
ALPHA Register							
8	7	6	5	4	3	2	1
A3	***A2***	***A1***	***A0***	***:***	***D2***	***D1***	***D0***

Digits 8, 7, 6 and 5 contain the address to be read or written. Leading zeros must be present. Values can range from 0x0000 through 0xFFFF, inclusive.

Digit 4 must be a colon, used as a field separator.

Digits 3, 2 and 1 contains the read or write data. In the case of a read the data in these digits is ignored and replaced with the actual read data by the function. Leading zeros must be present. Values can range from 0x000 through 0x3FF, inclusive. The limited range is because instruction memory is only ten bits wide.

The figure below shows an example of address 0xABCD and data 0x321.



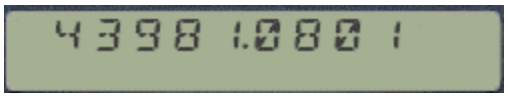
To use a decimal address and data the X-register is employed. The address field is to the left of the decimal point and the data field is to the right of the decimal point.

Leading zeros are not required for the address field, which can range from 0 through 65535, inclusive.

Leading zeros are required for the data field to make it four digits. The data field can range

from 0 through 1023, inclusive. Trailing zero digits can be omitted.

The figure below shows the decimal equivalent of the hexadecimal address/data pair shown above:



Converting between the two representations is supported by two dedicated functions, and manipulating the address or data is easy with the decimal representation.

AH>XD (Address and Data in ALPHA)

The **AH>XD** (*Hex Address/Data to Decimal Address/Data*) function takes the properly formatted address and data in the ALPHA register and converts it to the equivalent decimal address and data in the X-register. The stack is lifted prior to writing the X-register and the ALPHA register is unaffected.

AHPEEK (Address and Data in ALPHA)

The **AHPEEK** (*Instruction Memory Read using Hex Address/Data*) function reads the instruction memory specified by the address field in the ALPHA register and replaces the data field in the ALPHA register with the actual read data.

AHPOKE (Address and Data in ALPHA)

The **AHPOKE** (*Instruction Memory Write using Hex Address/Data*) function writes the instruction memory specified by the address field in the ALPHA register with the data field in the ALPHA register. The ALPHA register is unaffected.

XD>AH (Address and Data in X-register)

The **XD>AH** (*Decimal Address/Data to Hex Address/Data*) function takes the properly formatted address and data in the X-register and converts it to the equivalent hexadecimal address and data in the ALPHA register. The X-register is unaffected.

XDPEEK (Address and Data in X-register)

The **XDPEEK** (*Instruction Memory Read using Decimal Address/Data*) function reads the instruction memory specified by the address field in the X-register and replaces the data field in the X-register with the actual read data.

XDPOKE (Address and Data in X-register)

The **XDPOKE** (*Instruction Memory Write using Decimal Address/Data*) function writes the instruction memory specified by the address field in the X-register with the data field in the X-register. The X-register is unaffected.

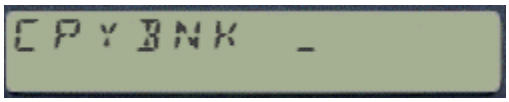
Code Copy Function

The Code Copy function provides a way to fill QROM pages with code from another source, usually a physical module.

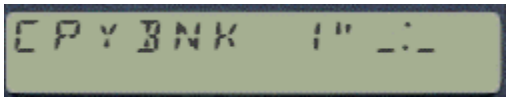
CPYBNK (prompts for Bank, SRC and DST)

The **CPYBNK** (*Copy Code Bank*) function copies an entire page of HP-41 Instruction memory to another page of Instruction memory. Although the function allows any bank of the source page to be specified, only transfers to Bank 1 of the destination page are supported. So specifying Bank 1 for the source page is the same as an unbanked instruction page copy. This function is not programmable because of the prompting required.

This function first prompts for the bank, as shown below:



Once the bank is entered the display switches to allow entry of the source and destination page (source first):



It is not possible to correct entries, and pressing the ← key at any point during parameter entry will cancel the function.

When copying a banked image to QROM there is subtlety that users should be aware of. The QROM uses the control-per-Port functionality that is used for most physical modules. This means that if the source page address and the destination page address (for QROM) are in the same Port address space the destination bank will match that of the source page because both pages share the same Bank-control bits. In this specific instance the destination bank will thus match the source bank.

HP-IL Code Copy Functions

The final four functions require that an HP-IL module (82160A) be present in the calculator, along with a tape drive (82161A) or other mass storage device on the HP-IL loop.

All of these functions may result in error messages from software in the HP-IL module, and those error messages will not be addressed here. Refer to the 82160A manual for that information.

The 4kx10 bits of ROM are stored in a file with a size of 640 registers, which corresponds to 5120 bytes. The 10-bit instruction words are read and written as five bytes for each four instruction words. Two different algorithms for the translation are available. The translation algorithm cannot be reliably determined from the data in the file, so the user is responsible for naming files appropriately if both translation algorithms are used on the same mass storage medium.

The ERAMCO file format packs the two most-significant bits from each of the four ROM words into a single byte and then the remaining four bytes are just the eight least-significant bits of the four ROM words.

ROM address	10-bit instructions
n	a9a8a7a6a5a4a3a2a1a0
n+1	b9b8b7b6b5b4b3b2b1b0
n+2	c9c8c7c6c5c4c3c2c1c0
n+3	d9d8d7d6d5d4d3d2d1d0

HP-IL order	ERAMCO format bytes
m	d9d8c9c8b9b8a9a8
m+1	d7d6d5d4d3d2d1d0
m+2	c7c6c5c4c3c2c1c0
m+3	b7b6b5b4b3b2b1b0
m+4	a7a6a5a4a3a2a1a0

The HEPAX format merely samples the bit stream from the ROM into byte-wide pieces.

ROM address	10-bit instructions
n	a9a8a7a6a5a4a3a2a1a0
n+1	b9b8b7b6b5b4b3b2b1b0
n+2	c9c8c7c6c5c4c3c2c1c0
n+3	d9d8d7d6d5d4d3d2d1d0

HP-IL order	HEPAX format bytes
m	a9a8a7a6a5a4a3a2
m+1	a1a0b9b8b7b6b5b4
m+2	b3b2b1b0c9c8c7c6
m+3	c5c4c3c2c1c0d9d8
m+4	d7d6d5d4d3d2d1d0

Neither format is better than the other. Software for going from ROM words to bytes is easier for the HEPAX algorithm, but software for going from bytes to ROM words is easier for the ERAMCO algorithm.

Storing ROM data in either format is not compatible with any of the recognized (PR, DA, KE, ST or WA) HP-41 file types supported by the software in the HP-IL module. The original HEPAX software used a DA file type (number 13) for ROM data, and the MAXX module also uses that file type. The original Eramco software used a special file type (number 7) that is not recognized by the HP-IL module software.

Executing the **DIR** function (located in the HP-IL module) will result in a listing with the following format for ROM files:

```
Eramco:          NAME    ??,S        640
Hepax:           NAME    DA          640
```

The "??,S" signifies an unknown file type and the fact that the file is secured. Securing the file is automatic for unknown file types. The Hepax case shows the Data file type, unsecured. In both cases the file is 640 registers in length.

The file name can be up to seven characters and is held in the ALPHA register. If the ALPHA register contains more than seven characters only the first seven are used for the file name. Each file name on the mass storage medium must be unique.

Only transfers to or from Bank 1 are supported, and the user is responsible for choosing the proper source or destination page. Any page, from 0 to 15, is allowed by these functions.

No check that the selected page has been enabled for reads or writes is performed. The user is responsible for guaranteeing that the specified page is appropriately enabled. For transfers to the IL device this means that the source page must be enabled for reads. If the source page is not enabled for reads a file containing all zeros will be written. For transfers from the IL device the destination page must be enabled for writes. If this is not the case all data read from the IL device will be ignored.

RDEIL (page in X-register, filename in ALPHA register)

The **RDEIL** (*Read ERAMCO-format ROM Page from HP-IL*) function copies an entire page of HP-41 Instruction memory from a mass storage device using HP-IL. The file is decoded using the ERAMCO algorithm.

RDHIL (page in X-register, filename in ALPHA register)

The **RDHIL** (*Read HEPAX-format ROM Page from HP-IL*) function copies an entire page of HP-41 Instruction memory from a mass storage device using HP-IL. The file is decoded using the HEPAX algorithm.

WREIL (page in X-register, filename in ALPHA register)

The **WREIL** (*Write ERAMCO-format ROM Page to HP-IL*) function copies an entire page of HP-41 Instruction memory to a mass storage device using HP-IL. The file is encoded using the ERAMCO algorithm.

WRHIL (page in X-register, filename in ALPHA register)

The **WRHIL** (*Write HEPAX-format ROM Page to HP-IL*) function copies an entire page of HP-41 Instruction memory to a mass storage device using HP-IL. The file is encoded using the HEPAX algorithm.

Error Messages

Error Message	Function	Meaning
ALPHA DATA	Functions requiring numeric data in the X-register	X-register contains Alpha data
DATA ERROR	Instruction Memory functions	Invalid hexadecimal
	YM Block functions YM Register functions	Range Error
	QROM functions, HP-IL Copy functions	Invalid page number
NO BANK	**CPYBNK**	No bank found
NO HPIL	HP-IL Copy functions	No 82160A detected
NONEXISTENT	YM Block functions	Invalid Block address
NOT OK	YM Block functions	Needs confirmation string **OK** in ALPHA register before modifying YM Block 0
OUT OF RANGE	Functions requiring a YM register address	Nonexistent register
	Instruction Memory functions	Address or data too large
TRANSMIT ERR	HP-IL Copy functions	Problem with HP-IL
TRY AGAIN	QROM functions	Page is currently occupied

Function XROM Numbers

References to functions are stored in HP-41 programs as XROM numbers. This technique is not visible to the user as long as the module containing the function is present in the calculator, because the HP-41 software automatically looks up the function name and displays this name in a program line. If the module is removed the XROM numbers become visible in program lines. The XROM numbers for the HP-41 MAXX module are listed below.

Function	XROM	Note
-MAXX 4C	15,00	Not programmable
ST>YM	15,01	
ST<>YM	15,02	
YM>ST	15,03	
MM>YM	15,04	
MM<>YM	15,05	
YM>MM	15,06	
XM>YM	15,07	
XM<>YM	15,08	
YM>XM	15,09	
YM<>YM	15,10	
YMCLR	15,11	
YARC	15,12	
YAST	15,13	
YDSE	15,14	
YISG	15,15	
YRCL	15,16	
YRC+	15,17	
YRC-	15,18	
YRC*	15,19	
YRC/	15,20	
YSTO	15,21	
YST+	15,22	
YST-	15,23	
YST*	15,24	
YST/	15,25	
YVEW	15,26	
YX<>	15,27	
YRGMOV	15,28	
YRGSWP	15,29	
CLYRG	15,30	
CLYRGX	15,31	
A<>YRG	15,32	
ST<>YRG	15,33	
QRINI	15,34	
QRCLR	15,35	
QRABY	15,36	
QRRE	15,37	
QRRO	15,38	
QRRP	15,39	
QRRWD	15,40	

Function	XROM	Note
QRRWE	15,41	
QRWE	15,42	
QRWO	15,43	
QRWP	15,44	
AHPEEK	15,45	
AHPOKE	15,46	
XDPEEK	15,47	
XDPOKE	15,48	
AH>XD	15,49	
XD>AH	15,50	
CPYBNK	15,51	Not programmable
WREIL	15,52	
WRHIL	15,53	
RDEIL	15,54	
RDHIL	15,55	
MXST?	15,56	
QRST?	15,57	

QROM Loading Examples

Loading QROM with software might seem like a daunting task, but the examples below should help to illustrate the required steps. All of these examples assume that the QROM programming starts out in the default state of page 15, unbanked, and disabled for reads and writes.

The "LDAXY" program loads a page from HP-IL. It assumes the IL file name in the ALPHA register, the destination *page.bank* in the Y-register and the QROM block number in the X-register.

```
01  LBL "LDAXY"    ; assumes IL file name in ALPHA, page.bank in Y and block# in X
02  AVIEW           ; display file name
03  QRWE            ; enable the block for writes
04  QRABY           ; set page and bank
05  X<>Y            ; destination page to X
06  RDEIL           ; read page (Eramco format)
07  X<>Y            ; block number to X
08  QRRO            ; enable page for read-only access
09  CLD
10  END
```

The "LDLIB" program loads Library-4 from HP-IL. It assumes the IL file names are LIBX1 and LIBX2 and uses QROM blocks 1 and 2 to hold the images. It is always safer to load secondary banks prior to loading Bank 1.

```
01  LBL "LDLIB"    ; assumes IL file names LIBX1 and LIBX2
02  4.2             ; page 4, bank 2
03  ENTER^
04  2               ; QROM block #2
05  "LIBX2"
06  XEQ "LDAXY"
07  4.1             ; page 4, bank 1
08  ENTER^
09  1               ; QROM block #1
10  "LIBX1"
10  XEQ "LDAXY"
11  TONE 7
11  END
```


The procedure below copies the contents of the HP-41 Advantage module (assumed to be inserted in Port 4) into QROM, using blocks 10-12. The procedure cannot be programmed because the CPYBNK function is not programmable. Copying the contents of a physical module into QROM frees up a physical port and also allows editing the module contents.

Because of the way that bank-switching works, the QROM blocks cannot be located in the same port as the physical module during the copy operation. This example will use Port 3 for the location of the QROM blocks.

The HP-41 Advantage module contains an unbanked 4K in the lower half of the Port and two banks in the upper half of the Port. The CPYBNK function only copies images to Bank 1, so the QROM block holding the Bank 2 portion of the Advantage mode must be relocated after the copy is complete.

```
13.1
ENTER^
12
QRABY           ; Assign QROM block #12 to page 13, bank 1
QRWO            ; and enable it for writes only
CPYBNK 2" F:D   ; copy the Advantage upper page bank 2
13.2
ENTER^
12
QRABY           ; Assign QROM block #12 to page 13, bank 2
13.1
ENTER^
11
QRABY           ; Assign QROM block #11 to page 13, bank 1
QRWO            ; and enable it for writes only
CPYBNK 1" F:D   ; copy the Advantage upper page bank 1
12.0
ENTER^
10
QRABY           ; Assign QROM block #10 to page 12, unbanked
QRWO            ; and enable it for writes only
CPYBNK 1" E:C   ; copy the Advantage lower page
```

Power off, remove the physical module. Power back on and execute the following to activate the internal copy of the Advantage module.

```
12
QRRO            ; enable bank 2 first
11
QRRO            ; then the upper page
10
QRRO            ; and finally, the main page
```


Internal Details

The MAXX Module appears to the MCODE programmer as a set of peripheral registers that can be accessed once the correct peripheral address of 0xF3 is selected.

It is possible to use the **AHPEEK** and **AHPOKE** functions to directly access these internal I/O registers. If you think that you need to use this feature please contact Systemyde for instructions on how to do it.

The table below shows the peripheral registers in page 0xF3 in the MAXX Module that are visible to the MCODE programmer. Unimplemented registers always return all zeros on read, and writes to these unimplemented registers are ignored.

Register	Usage	Read/Write	comments
0	Hardware Control	yes	digits 7:0 only
1	QROM Control A	yes	digits 11:0 only
2	QROM Control B	yes	digits 11:0 only
3	QROM Control C	yes	digits 11:0 only
4	Scratch 0	yes	
5	Scratch 1	yes	
6	Scratch 2	yes	
7	Scratch 3	yes	
8	Translate Input	yes	
9	Five Bytes	read-only	digits 9:0 only
10	Four Words	read-only	digits 11:0 only
11	ROM Control	yes	digit 0 only
14	Indirect Data	yes	
15	Indirect Pointer	yes	digit 0 only

The bit assignments in the control registers have been optimized for use with HP-41 mcode. For example, writing a "0" to a nibble in a control register has no effect. This allows only selected nibbles to be affected by a write, reducing the need to save and restore these registers in software. In cases where a control register needs to be saved and restored the four Scratch registers will save on required CPU resources.

Registers 9 and 10 are read-only packed and unpacked versions of the Translate Input register.

Register 11 provides read/write control for the block of memory that contains the MAXX functions.

Registers 14 and 15 implement indirect access for registers 0-11. The register address is loaded to the Indirect Pointer and then the addressed register can be read or written using the Indirect Data address. This simplifies software in some cases.

Hardware Control Register

Register 0 is the Hardware Control register. Only digits 7-0 are used.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	**tmr**	**xm2**	**xm1**	**xm0**	**mm3**	**mm2**	**mm1**	**mm0**

Nibbles 7 through 0 enable and disable certain hardware features in the MAXX module. These nibbles also report the status of these hardware features when read. The tables below show the valid bit combinations for these nibbles. No other bit patterns will ever be returned when reading this register.

bit pattern	Write Meaning
xx0x	No Change
xx10	Disable
xx11	Enable

bit pattern	Read Meaning
0000	Disabled
1111	Enabled

Nibble 7 (**tmr**) controls the Time module portion of the MAXX module, both the timer hardware and the Time module software.

Nibble 6 (**xm2**) controls the second Extended Memory module portion of the MAXX module, which corresponds to register addresses 0x300 through 0x3FF.

Nibble 5 (**xm1**) controls the first Extended Memory module portion of the MAXX module, which corresponds to register addresses 0x200 through 0x2FF.

Nibble 4 (**xm0**) controls the Extended Functions/Memory module portion of the MAXX module, both the software and the memory. The software occupies the lower half of the Port where the MAXX module is inserted, while the Extended memory corresponds to register addresses 0x040 through 0x0BF.

Nibbles 3-0 (**mm3**-**mm0**) control the regular Memory module portion of the MAXX module according to the table below.

nibble	Register address range
mm3	0x1C0-0x1FF
mm2	0x180-0x1BF
mm1	0x140-0x17F
mm0	0x100-0x13F

QROM Control Registers

Registers 1 through 3 are the QROM Control registers, with one control register for each four pages of QROM memory. The three groups of QROM are labelled A, B and C. Within each QROM group the individual blocks are numbered 1 through 4 as far as the control fields are concerned, but this numbering should not be confused with the operating mode for each block. Block 1 is controlled by nibbles 0, 4 and 8, block 2 is controlled by nibbles 1, 5 and 9, and so on.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	**q4pg**	**q3pg**	**q2pg**	**q1pg**	**q4b**	**q3b**	**q2b**	**q1b**	**q4m**	**q3m**	**q2m**	**q1m**

Nibbles 11-8 (**qxpg**) select the address where each block of QROM will reside. QROM blocks can only be assigned to pages 4 or 6 through 15.

The QROM page addresses should be set before enabling the blocks for reads. These nibbles also report the status of the QROM blocks when the control register is read. The tables below show the valid bit combinations for these nibbles. No other bit patterns will ever be returned when reading this register.

qxpg bit pattern	Write Meaning
00xx	No effect
0100	Page 4
0101	No effect
0110	Page 6
0111	Page 7
1000	Page 8
1001	Page 9
1010	Page 10
1011	Page 11
1100	Page 12
1101	Page 13
1110	Page 14
1111	Page 15

qxpg bit pattern	Read Meaning
0100	Page 4
0110	Page 6
0111	Page 7
1000	Page 8
1001	Page 9
1010	Page 10
1011	Page 11
1100	Page 12
1101	Page 13
1110	Page 14
1111	Page 15

Nibbles 7-4 (**q*x*b**) control the banking operation of the QROM blocks. These nibbles also report the status of the QROM blocks when read. The tables below show the valid bit combinations for these nibbles. No other bit patterns will ever be returned when reading this register.

q*x*b bit pattern	**Write Meaning**
00xx	No effect
0100	Bank 1
0101	Bank 3
0110	Bank 2
0111	Bank 4
10xx	Not Banked
11xx	No effect

q*x*b bit pattern	**Read Meaning**
0100	Bank 1
0101	Bank 3
0110	Bank 2
0111	Bank 4
1000	Not banked

The odd encoding for the bank select is carried over from the ENROM*x* instructions that control bank switching. It is easier if everything uses the same encoding. This field also allows the block to be programmed to ignore the bank control bits.

Nibbles 3-0 (**q*x*m**) control the read and write operation of the QROM blocks. These nibbles also report the status of the blocks when read. The tables below show the valid bit combinations for this nibble. No other bit patterns will ever be returned when reading this register.

q*x*m bit pattern	**Write Meaning**
xx0x	No Read Change
xx10	Read Disable
xx11	Read Enable
0xxx	No Write Change
10xx	Write Disable
11xx	Write Enable

q*x*m bit pattern	**Read Meaning**
0000	Read & Write Disabled
0011	Only Read Enabled
1100	Only Write Enabled
1111	Read & Write Enabled

Scratch *x* Register

Registers 4-7 are the Scratch 0-3 registers. These registers are available for any software use, but are used in the MAXX functions for saving and restoring the state of the control registers to reduce software overhead.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
s13	s12	s11	s10	s9	s8	s7	s6	s5	s4	s3	s2	s1	s0

Translate Input Register

Register 8 is the Translate Input register. This read-write register is the source for the packing/unpacking operation required for the byte-oriented HP-IL functions.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
m	**0**	**0**	**0**	**byte 4**		**byte 3**		**byte 2**		**byte 1**		**byte 0**	
m	**0**	**word 0**				**word 1**				**word 2**			**word 3**

Nibble 13 (**m**) controls the algorithm selection for the pack/unpack operation. A 0 in this nibble selects the ERAMCO algorithm, while a 1 selects the HEPAX algorithm.

When unpacking, nibbles 9-0 (**byte *x***) is the data to be unpacked. Order is important. The first received HP-IL byte is **byte 0** and the last received HP-IL byte is **byte 4**.

When packing, nibbles 11-0 (**word *x***) is the data to be packed. Order is important. The lowest ROM address word is **word 0** and the highest ROM address word is **word 3**.

Five Bytes Register

Register 9 is the Five Bytes register. This read-only register returns the contents of the Translate Input register, packed using either the ERAMCO or HEPAX algorithm. Only digits 9-0 are used. The packed contents are not available until one instruction time after the Translate Input register is written.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**byte 4**		**byte 3**		**byte 2**		**byte 1**		**byte 0**	

Four Words Register

Register 10 is the Four Words register. This read-only register returns the contents of the Translate Input register, unpacked using either the ERAMCO or HEPAX algorithm. Only digits 11-0 are used. The unpacked contents are not available until one instruction time after the Translate Input register is written.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**word 0**				**word 1**				**word 2**			**word 3**

ROM Control Register

Register 11 is the ROM Control register, which provides a write enable for the block of instruction memory that contains the MAXX software. Only digit 0 is used.

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

Nibble 0 (**rwe**) controls the write operation for the block. This nibble also reports the status of the block when read. A bit combination of 0xF enables writes, and any other bit combination disables writes.

Indirect Data Register

Register 14 allows software to indirectly address any of the other MAXX registers. The register address is written to the Indirect Address register and then the selected register can be read and written via this register.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
d13	d12	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0

Indirect Address Register

Register 15 contains the register address for the register to be accessed.

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

Nibble 0 (**ia**) is the address of the register to be accessed indirectly via the Indirect Data register. This addressing uses the same register addressing that was shown earlier. Obviously, accessing the Indirect Data register itself does not make sense, and all zeros will be read and writes will be ignored in this case.

Expanded Register Access

The processor used in the 41C series has native support for 4096 registers, via a twelve bit register address. This register address is loaded into memory devices using the DADD=C instruction. However, the memory devices used in the calculator only latch and decode ten bits of this register address. This means that a different mechanism is required to access registers beyond this ten-bit address limit to avoid any conflict with memory devices already present in the 41C.

Borrowing from the technique used for peripheral registers, with the PFAD=C instruction, the MAXX hardware decodes a new instruction called EADD=C. This Extended Address load instruction loads all twelve bits of the register address for use by the MAXX hardware. Since the 41C CPU and memory devices do not recognize this instruction there will be no decoding conflict as long as the regular DADD=C instruction has previously stored an unimplemeted register address. This is exactly how decoding conflicts with peripheral registers are avoided in the 41C.

The EADD=C instruction uses the bit pattern 0x0C0, and a typical expanded register access will involve the following instruction sequence:

```
        LDI             010             ; empty 41C register address
        DADD=C                          ; select nonexistent 41C register
        LDI             200
        C=C+C           X               ; expanded register 400
        EADD=C                          ; load expanded register address
```

As in the case of the PFAD=C instruction, any subsequent DADD=C instruction will clear the address latched by the EADD=C instruction, returning the MAXX hardware to normal 41C operation, making the expanded register access completely transparent to the existing 41C hardware and software.

Shown below is an actual code snippet from the MAXX functions. In this code the register address, which may be in either data memory or Expanded memory, is held in A.X. The type of address (0=data, 1=Expanded) is held in A.S and the data to be written is held in the N register. This code writes the data after first setting the register address appropriately.

```
(DST?)          LDI             010             ; C = xxxxxxxxxxx010
                DADD=C                          ;                       disable ram 0
                C=A             X               ; C = xxxxxxxxxxxddd
                ?A#0            S               ;                       dst loc?
                GONC            (DST_W0)
                #0C0                            ; EADD=C                enable ram x
                GONC            (DST_WR)
(DST_W0)        DADD=C                          ;                       enable ram 0
(DST_WR)        C=N                             ; C = d-data-ddddddd
                DATA=C                          ;                       write dst data
```


Revision History

05/04/2022	Preliminary release.
06/06/2022	Miscellaneous edits, added Error Messages and Internal Details.
07/23/2022	Miscellaneous edits, added YRFNDX function
08/19/2022	More figures, Split mode for Banked memory
09/26/2022	Changes all over the place, for version -1B
10/24/2022	Fixed a formatting problem; clarified collision checking.
10/29/2022	Figures on pages 13 and 14 were reversed.
11/19/2022	Changes all over the place, for version -2A and Instruction RAM
11/21/2022	Instruction RAM control register details
12/06/2022	Change "Instruction RAM" name to "QROM"
12/15/2022	Add explanation about hardware read and write prioritization
02/09/2023	Changes for revision -3A. Modifications to QROM functions, as well as HP-IL Code Copy functions and Internal Details.
02/23/2023	Various edits. Changed QROFF to QRRWD and QRON to QRRWE. Added QROM Loading Examples section.
03/10/2023	Changes for -3B. Changed HP-IL Copy function names to free up a few bytes of code space. HP-IL file format change to Hepax case for interoperability.
03/31/2023	Changes for -4B, plus edits based on reviewer feedback.
06/30/2023	Changes for -4C: clarification for QROM initialization, Auto-configure.
09/01/2023	Added "READ THIS FIRST" section, plus various edits throughout.
09/04/2023	Added "Function XROM Numbers" section, plus various edits.
11/26.2023	Added to "READ THIS FIRST" section
12/03/2023	Fixed a problem with the example in Expanded Memory Access. Thank you, Meindert!

