ERAMCO
SYSTEMS

ES 83120A
ERAMCO MLDL-OPERATING SYSTEM EPROM

OWNER'S MANUAL

February 1984

Printed in the Netherlands

CONTENTS

INTRODUCTION

This manual deals with the **ERAMCO MLDL** operating system eprom. To get a full understanding of all the routines and functions in this eprom set, it is advisable to read through this manual carefully before operating any of the functions or routines.

INSTALLATION

Follow the instructions of your **ERAMCO MLDL-box** carefully when installing the eprom set in your box. It may be necessary to bend the feet of the two eproms slightly inward to make them fit easily into the eprom sockets. Do not forget to enable the page on which you insert the eproms (for more detailed information on how to insert the eproms, consult your hardware manual of the **ERAMCO MLDL-box**). A lower address is the most appropiate page for insertion of the eprom. This provides a quick access to the routines and functions available in the **ERAMCO MLDL**-eprom set.

ORGANISATION OF THE INSTRUCTION SET

As you will soon discover the routines and functions in this eprom set are divided into three sections. The first section contains all the functions and routines that will change anything in the **MLDL**-ram you are working on. So always be careful when using any of these functions. A single mistake can destroy the whole 4K ram block that is under development.

The second section contains the functions that facilitates working with the **MLDL**-ram. They do not change anything in the ram but will provide a quicker access to the ram (LROM will tell you almost immediately where you can continue with writing in the ram or where you can store a User-code program).

The third and last section in fact belongs to the two mentioned above. However, this is a seperate section to keep compatible with the xrom numbers of an older version.

Note : All inputs which has to be placed in the alpha-register are related to hexadecimal

In the description of the functions it is assumed, that you have one **MLDL** ram page available for exercising the examples. To ensure that the examples work out in the way we have described them, is it necessary to clear one block and to place it at the proper page. Place the first block off your **MLDL** ram at page 7. This is easily achieved by turning the appropriate (left) hex rotary switch to 7. Disable the block by switching the left enable switch down (off). To avoid problems with the second block, it is advisable to switch this block of too.

After these preparations we can clear the whole block. Input for this is 7 in ALPHA. Now execute the function CLBL. For detailed information of it's operation see page 14.
Switch the **MLDL** ram page on line by switching the left enable switch to the ON state. It is now ready for the examples.

INPUT : All the hexadecimal input in the ALPHA register is checked on valid data. Data is valid only, if it consists of the hexadecimal characters. These characters are the numbers from 0 upto 9 and the letters A through F. Any other character in ALPHA will cause an error. The display will show DATA ERROR.
If the error occurs in a function during a running program, the error will be displayed and the program is halted at the instruction, that caused the error.

OUTPUT : Every function in this **MLDL** rom that gives an hexadecimal output to the ALPHA register, will automaticcally execute an AVIEW after it has put it's data into the ALPHA register. So, if you are using for example the function LOCA in a program, it is not necessary to do a AVIEW after the function. (Otherwise the result will be displayed twice. In conjunction with the printer your results will also be printed twice.

MLDL WRITE FUNCTIONS

RAMWR (RAM WRite)
XROM 11,Ø1

This non programmable function allows the user to read every word in a ROM, EPROM, or **MLDL**-ram (EROM). In case of **MLDL**-ram it is also possible to change or write in this **MLDL**-ram. The addresses and data are prompted for and given in hexadecimal form. This function will redefine the keyboard as long as it is used to make hexadecimal input easier.

After calling this function it will prompt for the absolute address in ROM. The following keys are now active: Ø-9, A-F, back-arrow and the on key. The back-arrow key is used in the usual way to correct the last given input. NULL will be displayed if you hold the last input-key. When you release this key after NULL is displayed, you will be prompted again for the address. Pressing back-arrow without input causes the function to exit to normal operation of the HP-41.

The address and three prompt signs are shown in the display (AAAA ___). From now on the keyboard is defined as follows:

-STO will give you the data at this and the following addresses. Each address and the data are displayed for about Ø.5 sec. Pressing any key accept the R/S or the ON key, will slow down the listing of the data that is displayed. The R/S key will stop the listing at any desired place. The ON key will switch off the machine in the usual way.

Example : If you press RAMWR and fill in the prompt with XFD5 (X represents the page the **MLDL** rom is located) you will see Ø93. This is the last letter of the xrom name of the **MLDL** rom. if you press STO, you will see the whole name of the rom, displayed one character at the time. Stop the display after you have seen 3EØ. This is the end of the xrom name.

-TAN or BST decreases the address by one. This enables you to go through the listing by hand.

Example : After you have stopped the listing in the previous example, you can see the first letter of the xrom name, by pressing TAN or BST once. The display shows XFFD 005.

-SST increases the address by one, making it possible to step through the listing by hand.

Example : Pressing SST once places you at the end of the xrom name. The display shows XFFE 3E0. Pressing SST once more places you at address XFDF with data 092.

-back-arrow asks you for a new address if there is no data input. Otherwise it will operate in the usual way to correct the last input.

Example : Press back-arrow once. You are prompted for the new address. Fill in the prompt with 2FFE. This address contains the revision level of the second operating system rom. The number represents the position of the letter in the alphabet. So if you see 006, your revision level is F.

-"0", "1", "2", "3" (numberkey's 0,1,2,3) are interpreted as new data. In this way wrong data input is prevented, because the first character of a data word can only be 0,1,2 or 3. For the rest of the data input the hexadecimal keyboard is available again. Holding the last data key will NULL the input function and after releasing the key will prompt for new data. With the back-arrow key it is possible to correct the last given input. The address will be increased by one after completion of data input. This will facilitate the writing of long programs.

Example : We will initialize our ram block with a name. Therefore we have to go to page 7. Press back-arrow once and fill in the prompt with 7000. At this address the XROM number of our rom is located, and we have to give the ram block an XROM number before writing to it. This is necessary, for RAMWR checks this address every time we write to ram. If it is zero, the message NO ROM is given and we are asked for an address again.
The XROM number we will use is 31. This is the same XROM number as the cardreader, so to avoid problems you should disconnect your cardreader.
After this is done, we can start writing to our **MLDL** ram. Press back-arrow again and goto address 7086. The first thing to do, is to give the **MLDL** ram block a name. The name we are going to use is NEWUSER 01.
This name is coded as follows :

Address	Data	Comment
7086	0B1	1 end of the name
7087	030	0
7088	020	space
7089	012	R
708A	005	E
708B	013	S
708C	015	U
708D	017	W
708E	005	E
708F	00E	N start of the name
7090	3E0	start of function

The name can easily be entered by pressing the data words after each other. If you make a mistake during entry, you can correct it with the back-arrow key. If you discover the mistake after you have finished the data word, you can go back with BST or TAN and try it again. With AFAT we will complete the initialization of our **MLDL** ram page. Press backarrow twice to exit the RAMWR mode.

You can exit this function, when you are in input-mode, by means of pressing the back-arrow key twice.

If you are at address $0000 and you try to do a backstep, you will find yourself at $0001. This is done to avoid an unexpected wrap around to $FFFF. If you really want to backstep to $FFFF you have to press backarrow once and continue at this address.

WARNING : Be careful with the addresses from xFF4 up to xFFA. These addresses are scanned by the operating system of your calculator. It's possible that the calculator will crash when these adresses contain error data. For more information see appendix F.

MMTORAM (Main Memory TO RAM)
XROM 11,02

The function MMTORAM is used to copy a program from main memory in the calculator to the desired **MLDL**-ram page in a **MLDL**-box. All the necessary translations for a good operation of this program are made automatically. The Function Access Table (FAT) is updated at the same time with the new Global Labels of the program. For good operation of this function it is necessary to initialize the **MLDL**-ram in the proper way.

Preparation of the **MLDL**-ram: You need a block of ram words that is long enough to hold the desired program. The length of the program can be found with the help of CBT (see CBT). Add two to this number of bytes and you have the number of bytes that will be needed for the program when loaded into the **MLDL**-ram. Now you must find a block in the ram space that is large enough. Write down the starting address of this block. **BE CAREFUL** Addresses in ram are given in hexadecimal form, but the length of the program (by CBT) is given in decimal form. Key into ALPHA the starting address of the block (it's advisable to leave about 20 words between the starting address of the block where the program will be written and the first empty word in the ram you have found, for future revisions).

When you are initializing a 4K block of **MLDL** ram automatically with the help of IPAGE, you do not have to do all of this. The loading address will be automatically given by IPAGE. Also the first next empty word will be returned by MMTORAM to the ALPHA register, to make loading easier.

User flags 0 and 1 can be set or cleared to achieve the desired private status

flag 0	flag 1	status
cleared	cleared	program open
cleared	set	program open, after COPY private
set	cleared	program private
set	set	program private

With the help of these two user flags it is possible to make the program completely private in the **MLDL**-ram, e.g. you can not go into PRGM mode to examine the program and it is not possible to copy the program into the main memory with the help of the COPY function. A partly private status is also possible. In this case it is possible to examine the program, but after copying it into the main memory it will be private. The third option means no security at all. Programs are now free to examine and to copy (compare with e.g. the math module).Please note that changes in the program are only possible when it is stored in main memory (see the manual of the calculator for it's behavior when you are in rom).

With user flag 3 you can have the option to delete the numeric labels in a program. (for more information about this option see CMPDL).
When this flag is set, nothing unusual will happen. The program is first compiled and then loaded into **MLDL**-ram with the desired private status according to the settings of flag 0 and 1.
If this flag is cleared to the contrary, the program will be loaded with all numeric labels deleted. (if this is possible)

MMTORAM can be executed after these preperations regarding the user flags. The function will prompt for the name of the program that has to be copied. It is enough to press ALPHA twice when the program counter is already set in the wanted program. Otherwise you must enter the name of the program in the same way as with CLP or COPY.

MMTORAM calls one of the two present compilers, depending on the status of user flag 3 and will compile the program (for messages during compilation see COMPILE). When the program is compiled, the message LOADING PGM will be displayed. When the whole process is finished, a tone will sound and the message READY will be displayed.

When the function has been finished, it will return the address of the next free byte in **MLDL**-ram. Be carefull. If you are loading manually, this is the address of the first byte after the program. It doesn't have to be necessarily empty. Whenever you are loading, with the **MLDL**-page initialized with IPAGE, it will be the next free byte available.

A CAT 2 or a CAT x (x is the pagenumber of the **MLDL**-ram where the program has been written on) will show you the updated FAT with the new labels.
Noting down the start and end-address of the used block will allow you to make changes without address mistakes.

For an example of how to load your user code programs in the **MLDL** box, we rever to **How to set up your own EROM page.** There a complete description is given how to set up a **MLDL** ram page for loading user-code programs.

AFAT (Append FAT entry)
XROM 11,03

The function AFAT enables the user to update the FAT, e.g. to append the starting address of a routine that has been written in the **MLDL**-ram. Functions are only accessable to the HP-41 when they have an entry in the FAT. This also holds true for programs that are transferred to the **MLDL**-ram. The function MMTORAM takes care of this automaticaly.

Input for AFAT is in the format UOPAAA. AAA is the start-address of the function within a page, P is the page number where the function is loaded, O is an offset and U tells the HP-41 if the routine is a M-code routine or a User code program.

U=0 M-code function. The address points to the first word that is executable
U=2 User code routine. The address points to a Global Label

Example : AAA=3FF The start of the function or routine is found at X3FF.

In order to understand the interaction of O and P it is necessary to realise that EPROM and **MLDL**-ram can be placed at every wanted page, e.g. at any desired port. It must also be kept in mind that an EPROM or **MLDL**-ram page contains only 4K. The value of P is only pointing to the page where the **MLDL**-ram is positioned at at this moment. The value of P will change when you address the **MLDL**-ram to a different page. Opposite to this is the behavior of the value for O. O is a constant added to the pagenumber. It will not change when you place the **MLDL**-ram at a different page. The constant O allows you the possibility to execute functions and routines from another page other than the one where the FAT entry is lodged. So it is evident that the page which is called must always be O pages further in the memory.

Example : The page that contain the FAT is at page 8, and the page that contain the routine itself is at page C, address is 490. We want to make an entry for a User-code routine with AFAT.

The value of O (the offset) is C - 8 = 4
The value of P (page containing the fat) is 8.
The value of AAA (start-address) is 490.
The value of U (M- or User code) is 2.
We do now need the following input for AFAT

248490

When we move the first ROM to another address we must also move the second ROM the same number of pages in the same direction if the value of O is something else then zero. Leading zero's in the input can be omitted

Example : For our **MLDL** ram we have written the rom name with the help of RAMWR. To be able to see the rom name when we are executing a catalog 2, we have to place the xrom name entry into the FAT. This is done with AFAT.
We do have a function name, so the digit representing U will be zero.
The rom name is not located at another page, so the offset is also zero.
We are working at page 7, so the value of P will be 7.
The starting address of the function is the first executable word of the function and is in our case located at 090.

This results in a total entry for AFAT of 007090

As leading zero's can be omitted, we can use 7090 as the entry address for AFAT. Write the entry into ALPHA. Go out of ALPHA and execute AFAT. If you do now a catalog 2 you will see NEWUSER 01 in the display when the catalog routine has arrived at page 7. (if you have no printer or timer module, it will be the first name that appears in the catalog.

DFAT (Delete FAT entry)
XROM 11,04

The function DFAT is used when you want to delete an entry from the FAT. The function or routine which is deleted will be invisible for the HP-41 after execution of DFAT. The XROM numbers of all the routines and functions that came after the deleted function in the FAT, will get one lower. Pay attention to this fact when you use functions or routines from the ram you are working on. The same input format is used as with AFAT. The difference is that you do not need to specify the value of U.
So the input format will be OPAAA (offset),(page),(address).
DFAT will search in the page with number P and delete the specified entry. Leading zeros may be omitted.

Example : In the example of the function AFAT we have added the function name to the FAT, to give the **MLDL** ram page a name. We will add another name to the FAT, USER Ø1, by appending a name to the FAT with address 708D. (for detailed instructions how to append an entry to the FAT see AFAT).
If you execute a catalog 2, you will see NEWUSER Ø1 and after this USER Ø1. The last entry has to be removed.
This is easily accomplished by getting the right entry address into ALPHA and execution of DFAT.
Give in ALPHA the entry address of USER Ø1. This address is 708D. Get out of ALPHA and execute DFAT.
With a catalog 2 you can check, that the entry has been removed. You should only see NEWUSER Ø1 in the catalog.

MOVE (MOVE ram block)
XROM 11,Ø5

The function MOVE allows the user to move certain parts in a ROM, EPROM or **MLDL**-ram to another place. Keep in mind that you can only move into **MLDL**-ram. MOVE makes it possible to insert words or delete words at any place in the **MLDL**-ram. It is also advisable to copy only small routines or functions from another page to the **MLDL**-ram page you are working on.

The input format in ALPHA is as follows : BBBBEEEEDDDD

BBBB gives the starting address of the block that has to be moved (it is the first word that will be moved).

EEEE gives the end-address of the block that has to be moved (it is the last word that will be moved).

DDDD gives the address of the first word of the block where the source block will be copied.
The function will accept a destination address within the original block.

Example : We want to copy the rom name to another part of the rom, to be able to make some changes and to use it as a second header. This second name has to start at address 7DDE. The rom name is located at 7086 to 7090.

The begin address is 7086
The end address is 7090
The destination address is 7DDE

This gives a total entry for move of 708670907DDE. Enter this in ALPHA and execute MOVE.
With the help of RAMWR you can check, that the word at 7DDE is 0B1 and at 7DE8 is 3E0. These are the first and last words of the rom name.

CLBL (Clear ram Block)
XROM 11,06

Clearing a block of **MLDL**-ram is done with the help of CLBL. Input is in ALPHA in the format BBBBEEEE.
BBBB is the first word of the block that has to be cleared.
EEEE is the last word of the block that must be cleared.
Execution of CLBL puts zero in all the addresses between the given ones, including the start and end addresses.

Example : We discover after some time, that we don't want to use the second rom name after all. We could leave it in the ram page, but for good housekeeping we want it to be cleared. This is accomplished by getting the right begin and end address into ALPHA and execution of CLBL. Switch to ALPHA and give as input the start and end address of the block of code we created with MOVE. The starting address of this block is 7DDE (destination address when we moved). The end address is 7DE8 (this we have found with RAMWR).
So the total entry for CLBL is 7DDE7DE8. Get out of ALPHA and execute CLBL. With RAMWR you can check, that the words at the specified addresses are deleted.

Another option of CLBL is to clear a whole 4K block at once. For this input P in ALPHA. P represents the pagenumber of the page you want to clear. ****** ATTENTION ****** This last option is dangerous. It operates like MEMORY LOST, but in this case it is a memory loss of the specified **MLDL**-ram page.

Example : Switch the other page of **MLDL** ram to page 6. Get into ALPHA and give the address of the page to be cleared (6). Get out of ALPHA and execute CLBL. Now you can switch the second **MLDL** ram page on line by setting the right enable switch to the ON position.

COPYR (COPY Rom page)
XROM 11,07

The function COPYR enables the user to copy an entire page of ROM or **MLDL**-ram to another page of **MLDL**-ram. This gives you the opportunity to change anything you want in the just copied block of ROM.

Input is in ALPHA and has the format SD.
S is the page from where the copy has to be made (Source).
D is the page to which the copy is destined (Destination).

This function will sound a low tone to indicate the completion of the function.

Example : We want to make a copy of our working **MLDL** ram page. This could be done with move by giving as input 70007FFF6000. But this will take longer and asks for a more complicated input. Therefore we will make use of COPYR. The input for this example is 76 in ALPHA. When this is done, the function COPYR can be executed. After the tone has sounded we can check, if the second rom is available by executing a catalog 2. You will now see the romname NEWUSER 01 appearing twice in the catalog.

ROMSUM
XROM 11,08

To check if a ROM is still in good shape HEWLETT-PACKARD has put a checksum in each ROM. With the function ROMSUM you are able to compute this checksum and put it at the proper place in the **MLDL**-ram you are developing. The checksum is calculated by adding all the words on this page, take modulo 255 and put the remainder in xFFF.
The input is P in ALPHA. P is the page number of the **MLDL**-ram you want to update the checksum.

Example : To be able to detect if our rom is still in good shape, we are going to compute the checksum of the rom. Give the address of the rom in ALPHA. Attention, we are using the second **MLDL** ram page now, so the input will be 6 instead of 7. Get back to normal operation mode again and execute the function ROMSUM. This will take a few seconds. During this time the display will remain blank.
When the function is completed, you can check if the checksum is calculated in the proper way. This is achieved by keying into the X-register the used xrom number 31. Now execute ROMCHKX. The display will change into 31 @@-@@ TST. After a few seconds it will change to 31 @@-@@ OK.
(Remember 31 is the xrom number we used for our **MLDL** ram page).

REG>ROM (REGisters to ROM)
XROM 11,09

This function is the opposite of ROM>REG (for more information on this function see at ROM>REG). This routine will translate the registers with it's 5 words/register back into 5 different words and place them at the proper addresses in a **MLDL**-ram page.
The input in the Y-register determines where the data will be put back in the **MLDL**-ram. 3 different options are available to achieve this.

1. "Y"= 0 — The block will be placed at the same location as where the original was (if the original was located from 83FF to 8456 it will be restored at the same addresses.

2. "Y"= P — P represents a page number that is created with the help of COD. The block will now be loaded at the same relative addresses from which it came from but at a different page(if the original was located at 83FF to 8456 it will be restored at P3FF to P456).

3. Y = BBBB — Here BBBB represents the start-address where the block will be stored (BBBB >= 0010). The block will be loaded starting at the address given by BBBB independent from the original start-address of the block.

The X-register must hold the number of the register that contains the first data words of the block that has to be read back (actually the first register contains a header that is used by REG>ROM and is made by ROM>REG).
Writing entire 4K blocks of **MLDL**-ram from a storage medium is facilitated by the functions SAVEROM and GETROM.

Example : Let us assume, that you have used the function ROM>REG before. This can be accomplished by getting to the example of ROM>REG at page 23. Here the romname is loaded to the registers in order to save it on magnetic cards or a cassette drive.

First we will load the data back to it's original place. To see this really happening, we must first clear the block, where the data is located. This is done by CLBL. Put in ALPHA the begin and end address of the block to be cleared (70867090). Execute CLBL to remove the data from the **MLDL** ram page.

We can now restore the data by getting it back with REG>ROM.
First we are going to get it back to the original place in the **MLDL** ram page. This is necessary in order to get our rom-name back. Input for this is zero in register Y and zero in the X-register. The data will be loaded back at it's original place. You can check this with RAMWR.

We also want the data loaded back at a completely different page. Therefore it is needed to get the page number into the Y register This is accomplished with the function COD. Place in ALPHA the letter representing the page we want to store to (6). After getting out of ALPHA we execute COD. The display will change a little. Now press 0 to move the binairy representation of the page to the Y register and get the address of the header register in the X register. Now execute REG>ROM. You will find at the addresses 6086 to 6090 the data that also is located at 7086 to 7090.

The last option of REG>ROM is to restore the data at completely different addresses. If we don't want to have the data at address 7086, but at address 7AEE instead, we must make use of the last option of ROM>REG. Now we have to specify the starting address in the Y register. This is done as with the previous example. Place in ALPHA the starting address (7AEE) and execute COD. Again the display may differ from what you are used to. Press 0 to enter the starting address to Y and place the first register to use into the X register.
After these initial actions the function REG>ROM can be executed. After termination you can check with RAMWR to see if the data really got there.

XROM 11,10

This is not a normal function. It does not do anything when executed but it is used as a spacer from write routines and application routines within the **MLDL**-ram . One possible application is to use it as a NOP. It will also terminate data input without raising the stack.

UTILITY FUNCTIONS

COMPILE
XROM 11,11

The function COMPILE places in every numerical GTO and XEQ the distance to that numerical label. Programs prepared with the help of COMPILE will usually run faster than programs that have to calculate these distances while running. Two byte GOTO's that can not make the distance will be transformed to three byte GOTO's. Therefore your program can be made longer by this routine and it is required to have at least three registers left after the program. (.END. REG xxx with xxx not equal to zero).

Compile prompts for the name of the program you want to compile. Input is in the same way as with the mainframe function CLP. So if you are not in the program you want to compile, you must input the complete name. Otherwise it is possible to press ALPHA twice. The function will first pack the program (PACKING), then handle the two byte GOTO's (COMPL 2B G) and if needed (in this case compile has found a 2 byte GTO that can not make it and will replace it with a three byte GTO, thus causing insertion of null bytes that have to be packed as well) repeat this sequence. After this is done it will continue with the three byte's GOTO's and XEQ's (COMPL 3B G/X). After the routine is finished it will put the message READY in the display. Labels not found will give the error condition NO LBL xx, with the number xx as the label not found. When you switch to program mode you will find the program step that caused the error condition.

If the program has the .END. as last statement instead of a normal END, it will change the .END. into a normal one. This is done for MMTORAM, which expects a program to be terminated with a normal END.

To be able to change the .END. into a normal one, the compiler needs at least one empty register after the program. During the initial packing of the program a check is made to see if there is at least one register available. If this is not the case, the program will terminate with the message TRY AGAIN. If so you should decrease the number of allocated memory registers. (change size)

After execution of compile you will be placed at the first step of the program.

Deleting steps or adding steps in a program, will change the status of the program into a decompiled one. Reusing the compiler will speed up the execution after the editing session.

Example : Create the next program in your calculator

```
01 LBL 'TST
02 LBL 00
03 LBL 01
04 GTO 02
05 LBL 03
06 GTO 04
07 LBL 05
08 GTO 06
09 LBL 07
10 GTO 08
11 LBL 09
12 GTO 10
13 LBL 11
14 GTO 12
15 LBL 13
16 GTO 14
17 LBL 15
18 GTO 16
19 LBL 17
20 BEEP
21 GTO 00
22 LBL 02
23 GTO 03
24 LBL 04
25 GTO 05
26 LBL 06
27 GTO 07
28 LBL 08
29 GTO 09
30 LBL 10
31 GTO 11
32 LBL 12
33 GTO 13
34 LBL 14
35 GTO 15
36 LBL 16
37 GTO 17
```

If you execute this program after you have loaded it, you will notice the significant time it takes before you hear the first beep. You will hear the second one much sooner. Stop the program and goto step 1. Delete the superfluous label 01.
Execute the function COMPILE. You will be prompted for the name of the program to be compiled. Press ALPHA twice, since we are in the program already. (It's also possible to give the full name of the program (TST)).
Now the message PACKING is displayed. If you do not have enough room after the program, COMPILE will terminate with the message TRY AGAIN. Then the messages CMPL 2B G and CMPL 3B G/X will be showed shortly after each other. When the compiler is through these messages, a tone will be sounded and the display gives the message READY.

If you press PRGM once, you will find yourself at the start address of the program. Press PRGM once more and press R/S. Notify the fact that there is no delay before the first beep sounds.
Goto step one once more and delete label 00. Execution of COMPILE will give the error message NO LBL 00. If you go into PRGM mode you will be at the step that caused the error, step 19. Please restore the program with LBL 00 at step 01 again,because we are going to use this program again in the example of CMPDL.

LOCA (LOCAte word)
XROM 11,12

This function allows you to locate a data-word in a 4K block of ROM, EPROM or **MLDL**-ram.
The input format in ALPHA is as follows: BBBBDDD.
BBBB specifies the address from where LOCA starts searching in the 4K block. Actually it will start at BBBB + 1 to allow repeated search in the block. NONE will be displayed when the wanted data (DDD) is not found in this 4K block. Whenever a data-word is found, it will be displayed together with the address at which it is found. The data in ALPHA (adress + word) will be replaced with the data found.This makes it possible to continue searching for the same word.

Example : With a small user code program you can easily print out all the occurrences of an instruction in a rom or **MLDL** ram page. Create the following user code program (make sure you saved the TST program)

```
01 LBL 'LOCATE          05 AOFF
02 'ADD + DATA          06 LBL 01
03 AON                  07 LOCA
04 PROMPT               08 GTO 01
```

Input for this program could be a starting address like X000 and the data to search for could be 040. This would give you a complete list of all the **MLDL** WRITE instructions in the **MLDL** rom. Enter for X the page address where the **MLDL** rom is located (usually page F).

LROM (Last ROM word)
XROM 11,13

LROM searches backwards for the last non zero word in a block beginning at a given start-address. Input is AAAA in ALPHA. The display will give the address of the last non zero word and the value at this address. NONE will be returned when the block between the start address and the beginning of this 4K page does not contain any word (other than zero).
This function can be very useful when the end-address of the last program entered has to be found. In this case the easiest way is to put xFF4 into ALPHA and execute LROM. It will give you the address of the last word that is occupied by the program.

Example : If we want to find out where we can load our next user code programs, we could search for empty space with the help of RAMWR, but this would be rather cumbersome. To avoid this, we are going to use the function LROM. In this case we want to search on page 7, starting from the end and working backwards. Input for this is 7FFF in ALPHA. Execution of LROM will return 7AF73E0 to the display after a short search time. This tells us, that the next available word in our rom is at address 7AF8. If we are searching on a completely empty page, LROM will return the message NONE to the display, because it can not find any word unequal to zero on the page. Try this with page 5 for example. Input for this is 5FFF in ALPHA. Execute LROM. After a short while the message NONE will be displayed.

COD (CODe)
XROM 11,14

The hexadecimal number in the ALPHA-register is converted to it's -bit-representation and this will be placed in the X-register. The contents of the ALPHA-register is unchanged. The stack will be rolled up and the value in the X-register before COD was executed is placed in the LASTX-register.
The display won't be intelligable after the function COD has been executed. For the synthetic programmer this will sound normal.

Example : Input in ALPHA the hexadecimal address of our romname and the start address of our romname (70867090). Execute COD after placing the address in ALPHA. If we change the display format to fix 9, the display will look like this 0.0000708 90 Save this coded representation of the address, for we are using it to demonstrate an example with DECOD.
These so called non normalized numbers (NNN's) should not be used to make calculations, for they can hang up the calculator for quite some time. Also they can not be stored and recalled in the same mannner as normal numbers, for they are normalized after being recalled. This is easily demonstrated by pressing STO 01 and RCL 01 after each other. The result is a zero X register.

DECOD (DECODe)
XROM 11,15

The function DECOD is the opposite of the function COD. It will translate a -bit-representation in the X-register to the same hexadecimal form as is used by the function COD. The output is given in the ALPHA-register. When DECOD is executed manually DECOD will also give the hexedecimal representation in the display.

Example : We are going to use the same number as we have created with the function COD. First clear the ALPHA register. Now we must get back our just created number. If you do a RDN, it will come back to the X register. Execute the function DECOD. The hexadecimal representation of the number will appear in the display. If you press back-arrow once, it will disappear and the nonnormalized number is viewed again. Go into ALPHA and discover the hexadecimal representation here.

ROMCHKX (ROMCHeck by X-reg.)
XROM 11,16

This function enables you to check if a ROM or **MLDL**-ram is still in good shape. Important though is the fact that a ROM or **MLDL**-ram must contain a good computed checksum (see ROMSUM for the definition of the checksum). HP rom's will always contain a good checksum. During the test the XROM number is displayed along with the short form of the name and the revision number of the ROM. If the ROM or the **MLDL**-ram doesn't contain this short name or the revision number, the display will show @@-@@.

Input in the X-register, the XROM number of the ROM or **MLDL**-ram you want to test (an example is 30 for the cardreader). During the test XX NN-RR TST will be displayed. XX is the XROM number of the ROM that is tested, NN is the shortened name and RR is the revision number.

Output of ROMCHKX is the display XX NN-RR BAD (indicates a bad ROM) or the display XX NN-RR OK (indicates a good ROM) These outputs will be given only when the function is executed from the keyboard.

The behavior of ROMCHKX will be different when it is executed in a program; when a ROM is found to be good it will do the next step in the program. Else it will skip the next step (compare the function FS?: the rule do if true is in force).
When there is no ROM present with the desired XROM number the message NO ROM XX will be displayed. Again it's behavior in PRGM mode is different. It will act as if the ROM is bad and skip the next line.

Example : We can check if the **MLDL** operating system eprom is still good. For this we need an input of 11 in the X register (this is the xrom number of the **MLDL** rom). When we execute the function ROMCHKX, the display will change to 11 AS- A TST. This indicates that the rom with xrom number 11 is under test. The revision code of this rom is AS A. After a short time the display will change to 11 AS- A OK . When we execute ROMCHKX with a xrom number that is not present it will say NO ROM nn. This can be tried with zero in the X register because a rom never can have xrom nr 00. The display will show NO ROM 00 after ROMCHKX has been executed.

ROM>REG (ROM to REGisters)
XROM 11,17

All the credits for this function and its counterpart (REG>ROM) go to Paul Lind and Lynn Wilkins who have written these two routines. ROM>REG places 5 words of 10 bits each in one HP-41 register. To avoid damage to the stored data it is saved as alpha data. This guarantees an optimal use of the available registers in the main memory of the calculator. Because of these functions it is now possible to store the routines and functions that are written in a **MLDL**-ram on tape or cards and they make it easier to exchange M-code with other users.

To transfer complete blocks roms to and from tape the functions SAVEROM and GETROM are incorporated in this rom.

The input for this function must be given in the Y-register. It has the form BBBBEEEE.
BBBB is the address of the first word to store.
EEEE is the address of the last word to store.
This input has to be in binary and right justified. This is achieved by putting the BBBBEEEE form in ALPHA and executing COD after this. The binary representation can be transferred to the Y-register by means of keying in a number in the X-register. The X-register holds the number of the first data register that will be used as data store. (normally this will be register 00)
If the number of registers needed, exceeds the number of free registers you will get the error message NONEXISTENT.

There is also output from this function. In the LASTX-register the last used register is given. By subtracting X from LASTX you will get the number of used registers minus 1. If you add 1 to this you will get the number of registers needed to store the desired **MLDL**-ram block.

Example : We will save our romname in the user registers. This block of registers is also used for the example of the function REG>ROM. To execute this function properly, we have to give the block to be saved in a binary representation in the Y register. In the previous example we have already created the address in the ALPHA register, so we only have to execute the function COD. This gives us the binary representation of the block to be saved in the X register. We want to save the block in the user registers starting at register 00, so we have to enter zero into the X register. Press 0. This also moves the binary representation of the block to be saved to the Y register.
After these preparations the function ROM>REG can be executed. Pressing LASTX gives us the last used register. This means we needed 4 registers to store the block (3-0 + 1).

MNEM (MNEMonics)
XROM 11,18

This function will give in conjunction with DISASM the name of a M-code instruction that is fetched with DISASM. The mnemonics that are used are the so called HP-mnemonics (there are also PPC (Jacobs) mnemonics). The mnemonics are left as a string in the Z-register. Eventual surplus information (jump-distance, value, field specifications) is given in the T-register. In case of two word instructions the LASTX-register is used. The following User-code program makes it possible to translate every ROM that you want.

Example : With the following user code program you are able to print out the machine code on a rom page.

```
01 LBL 'mdis        Name of program
02 CLST             initialize the stack registers
03 STO L            initialize the LAST X-register
04 SF 21            makes program stop at aview
05 'start add?      ask for start-address
06 AON              make ready for input
07 PROMPT           ask and wait for input
08 AOFF             leave the ALPHA mode
09 COD              put the start-address in X
10 LBL 01           start of the loop
11 DISASM           get the instruction
12 AVIEW            view the address, value and the
                    character
13 MNEM             build the mnemonic in the stack
14 CLA              initialize the ALPHA-register
15 ARCL Z           get the first part of the mnemonic
16 '@               append a space
17 ARCL T           get the second part of the mnemonic
18 AVIEW            view the mnemonic
19 GTO 01           restart the loop
```

This routine is meant to be used in 'manual' mode. For use with the printer it must be rewritten. The choice is up to the user.

DISASM (DISASeMbler)
XROM 11,19

The function DISASM makes it possible to put the contents of ROM into the display. At the same time the character representation from the word is given in the display.

Input: The X-register must contain the address of the wanted word (this can be done with the help of COD).

Output: The X-register will be incremented by one to make it easy to use DISASM in a loop. The Y-register holds the binary value of the address and the data at this address (these values can be made visible with DECOD). The ALPHA-register contains AAAA WWW L

AAAA is the address of the wanted word.
WWW is the value of this word.
L   is the character representation of the word.

There are two ways to represent characters in the HP-41. One way is the use of the ASCII standard. The other way is derived from this standard by subtracting 40 [hex] from the codes in the range from 40 hex through 5F [hex]. This gives you codes that lay in the range from 0 hex to 1F [hex]. These are the codes, that are used for the display. Therefore DISASM will translate these codes to normal characters.

Example : To see how the function DISASM is used see the function MNEM and the related user code program to print the contents of a rom with microcode functions.

CAT (CATalog)
XROM 11,20

The function CAT gives you a selective CAT 2. This routine is especially useful when you have to examine the catalog of a ROM that is located at a higher numbered port. When the system is loaded with a lot of roms it will take a long time before you arrive at the desired ROM (maybe you must go through the TIMER, PRINTER, IL-MODULE before you reach the wanted ROM). The function prompts in the same way as the CAT function of the HP-41. The prompt can be answered with the hex digits 0-F (CAT will redefine the keyboard in the same way as RAMWR). Entering digits 0-3 results in the normal CAT function from the HP-41. Digits 5-F will start the catalog at the wanted page. For further details we refer to the manual of the HP-41.

Users of an HP-41CX have to be careful using this function. In some cases there have been crashes reported, due to changes in the functioning of the CAT function of the HP-41CX. This is highly dependant of the contents of the status registers.

Example : If the **MLDL** rom is installed at page F (this will usually be the case, when the box is delivered to you straight from the supplier) you would see with a normal CAT 2 all the functions of the roms that are physically located before the **MLDL** rom. At least one is located there at the moment, and that is the test rom, we are working on in our examples. So if you do a normal CAT 2 you will first see NEWUSER Ø1. To skip this part, we can start our catalog at page F. Execute the function CAT and fill the prompt with the digit F. The catalog will start up immediately at page F thus showing the contents of the **MLDL** rom.

CBT (Count BYtes)
XROM 11,21

This function counts the number of bytes that is occupied by a program. The END statement is taken in account. At the prompt the name of the desired program must be keyed in or if you are already in the desired program press ALPHA twice (compare with the function CLP).

Output is given in the display only. The stack and the ALPHA-register are left undisturbed.
If you try to get the length of a program that is resident in a rom module the error message ROM is given.

Example : At the explanation of COMPILE we have written a short user code program to demonstrate you the advantages of COMPILE. Execute COMPILE once more on this program to make sure the program is as compact as possible. Now you can find out how long the program actually is. If you execute CBT and press ALPHA twice, the display will change to 68 BYTES. This is the length of your program including the END statement
Remember this length for you will see that the use of CMPDL will significantly decrease the number of used bytes, thus giving you a lot of memory back.

SYNT (SYNTesize)
XROM 11,22

With this function you can create two- and some three bytes instructions in program memory without using the bytegrabber. Data for this function needs to be given in the X- and Y-register. The first byte of the instruction (decimal coded) is given in the X-register. The second byte is given in the Y-register. SYNT will place the instruction after the program line where the program counter is pointing at that moment. **ATTENTION :** this routine works both in PRGM and RUN mode. Therefore you must be very careful when assigning SYNT to a key. Carelessly pressing the assigned key will produce an unwanted line in your program or even worse.

Example : 159 ENTER^ 58 execute SYNT will give a TONE 8 in your program which is completely different from the normal TONE 8. An input of 247 in X and Y will give you a byte grabber.

GE (Go to .End)
XROM 11,23

This function is a sort of replacement of the GTO.. function of the HP-41. It will put you at the end of program memory, but it is not packing the memory. Furthermore it does not put an end to the last program in memory. When you do not know where you are in main memory use GE and you are at a familiar place again.
This routine will display 00 REG NNN and also circumvents the line number bug in the HP-41 operating system.

XROM 11,24

This is just a seperator for the second and third section. For more details see page 16.

UPDATE FUNCTIONS

SAVEROM
XROM 11,25

With this function you can save the contents of an entire rom on cassette tape. The input format for this function is a name in the alpha register and the desired page number in x.
A file will be created on tape of 640 registers, occupying 20 records.
Because there are a lot of users who have been using the Mountain Computer eprom burner set with the functions READROM and WRTROM we also included a user code program to be able to read back rom files in the old 824 format. This is the program 'RROM in appendix H.
The file identifier on tape for the new file created by SAVEROM is # 07. This means that the files are presented in the DIR as :

NAME ??,S 640

We have chosen for a nonexistant file type to be sure that the data is not accidently destroyed. Therefore the file is also automatically secured after creation. SAVEROM saves 7 records per file compared to WRTROM or 'WROM. Now you will be able to get the maximum number of roms on your tape (e.g. 24 files).

To get the maximum number of files on your tape it is recommended to do a NEWM with 27 file directory entry's. You can write 12 files on each side of the tape then. After having written 12 files you should protect the tape from rewinding from one side to the other by creating a dummyfile "ENDTAPE" of 300 registers.

Example : If you have a cassette drive you can try the following example. We will save the contents of our rom at page 7 on tape and read it back with GETROM. Give a filename in ALPHA, for example USER1. Since we have our rom at page 7, were also the HPIL module resides, we have to move it to another page. This could be page 5. If you can not use this page, place your rom at another page. If so replace in the following example the pagenumber with your new page number
We have the name in ALPHA and now we have to give the page address in the X register. In our example this will be 5. Execute SAVEROM. You will hear the cassette drive working for some time. If you watch the drive closely, you will notice that it writes 20 blocks after each other.
When the drive is ready again you could do a DIR and see as entry in the directory of the tape our just created romfile. It will be in the form as described under the function description,
e.g. USER1 ??,S 640.

GETROM
XROM 11,26

This is the counterpart of the SAVEROM function. Input format is the same, so the name must be in alpha and the page number must be in x. For more information on the format of the files, we refer to the function SAVEROM.
Getrom will read back the contents of the rom file and put it in the desired ram page. There is no checking done to see if the specified page is actually a ram page. This is to allow you to get a rom file to a page that is not switched on.

Example : If you have saved our rom file on tape, we can demonstrate it coming back. First of all clear the page we are working on. This is done with CLBL. You probably know by now how this function works, so it is left up to you to clear the block. Put in ALPHA the name of the file we want to read back, e.g. USER1. In the X register the page address should be entered to which we want the rom read back. In our case this will be page 5. Now the function GETROM can be executed. After it has finished, you can check if it is back again in the usual way with a CAT 2.

CMPDL
XROM 11,27

This is in fact nearly the same function as the normal COMPILE. Therefore we are refering to COMPILE for the set up of the flags and the input format for COMPILE. They are both equal.

The only difference is that this function will delete the numeric labels in the program while compiling. This shortens the program and speeds it up. This can be done, because the HP-41 remembers where to jump to in the jump and execute functions. So after the first run of a program, the HP-41 knows the distances to all the labels and will always jump this distance. It does not matter if there is a label or not. Therefore the labels can easily be deleted. Only when the program contains indirect jumps or xeq's is it impossible to do so. This is due to the fact, that the HP-41 can not remember all the possible addresses of all labels in the program. For this reason you can not use this function when the program contains a GTO ind or XEQ ind.

The program respects all the local labels. So the labels A through J and the labels a through e are respected and will not be deleted. This is necessary because the HP-41 searches for them when you use them from the keyboard.

When this function is executed, it will make use of the user registers to hold the addresses of the deleted labels. Therefore make sure that the number of allocated registers is more then the number of labels in the programs. If you don't take care of this the calculator might crash.

To protect the compiled status as much as possible we change the terminated by the .END. This protects you from accidently writing at the end of the program if you want to continue at the end of the programmemory with new programs.

During program compilation, you will see the following messages after each other.

```
PACKING
COMPL 2B G
COMPL 3B G/X
PACKING
COMPL 2B G
COMPL 3B G/X
READY
```

The compiler makes use of the normal compiler. First the whole program is compiled to find out where to jump to. Then all the LBL's are deleted and their addresses are remembered in the user registers. This is done during the packing stage. After this the program is compiled again. When the function is through you are at the beginning of the program.

The user registers contain the information where the program resided and where the specified labels in the program were. The structure of a register is as follows 10ØSSSSLLLLØNN. The first two digits indicate alpha type of data. The SSSS part gives you the start address of the program in program counter format. The LLLL part gives you the address of the label in the packed program without the labels. The NN part gives you the deleted label at this address.

Example : We will compile the program that we used by the example of COMPILE again. This time we are going to compile it with CMPDL. This is easily done. First make sure we have enough empty registers by setting the size to 18 or greater. We can now execute CMPDL. At the prompt give the name of the program : TST. After the compiler has finished we can see the results. Just run the program. Again there is no delay in the first beep. Also notify the fact that the flying goose does not move anymore. This is because the goose only moves one place to the right whenever the program encounters a label. But since all labels are deleted, it is not necessary anymore to move the goose. If you stop the program and execute the function CBT, you will get as result 48 BYTES. This implies that we have saved 20 bytes of memory, and in this case it means that the program is shortened by roughly one third of it's original length.

IPAGE
XROM 11,28

This function sets up a ram page to load user programs and/or assembler code functions. The entire specified page is cleared and the specified xrom number and the name in alpha are written at the appropriate places. This we have already done manually when we explained RAMWR and AFAT. With this function it will be much easier.

Input for this function in ALPHA is the name of the rom. This name must be from one to 11 characters. As it is the name of the rom it is advisable to make it at least 8 characters. This has two reasons. First, a function name of more then 7 characters can not be executed. Second and more important is the fact that the CAT function of the HP-41 CX searches for names that are longer then 7 characters. So, if you use a name of less then 8 characters, the rom name will not show up in the header catalog of the HP-41 CX. This is also the case with the CCD module, a module likely to spread out as much as the PPC rom. Second thing to give as input is the **MLDL** ram page number to be initialized. This page number is given in the X register. (in decimal)

When the function is executed, it will prompt you for the xrom number of the page. There is no checking done on the input, because it is possible to use other xrom numbers, but you can not execute a function in a rom with a xrom number higher then 31, so it is advisable to use a xrom number between 1 and 31. See for the already used xrom numbers appendix D.

The name that will be written to **MLDL** ram consists of the first eleven characters in the alpha register when you have no more then 12 characters. If you have more then 12 characters in alpha the name will be the first 11 characters that are left in the display after having it displayed. In other words the first 11 characters of the last 12 characters in the alpha register will be used and write into **MLDL** ram.
When you have less then 11 characters the last character can be an underscore.

Output of the function is in alpha the address of the first empty word as it is used for the function MMTORAM.

Example : We will now initialize our page with the help of IPAGE. First switch the **MLDL** ram page back from page 5 to page 7. Give the desired name in ALPHA. We will make use of the same name as we used in the examples before. It will be NEWUSER Ø1. Give the right page number in the X register (7). Now execute the function IPAGE. At the prompt the desired xrom number can be given. We will make use of xrom number 21. This is the xrom number for user roms. After a short while a tone will sound and the message READY will be in the display. Pressing ALPHA once gives you the first free byte available to load from. This will be address 7Ø92.

MKPR
XROM 11,29

This function allows you to make your programs private, even if you do not have a card reader. The function will respect the compiled status of the program. At the prompt you must fill in the name of the program that has to become private or if you want to make the current program private press alpha twice.

Example : If we want to secure our program compiled with CMPDL from accidently being altered we could make it private. Execute private and fill in the prompt with TST. If you switch to program mode you will now discover that the program is private.

APPENDIX A

XROM	NAME	INPUT	OUTPUT
11,01	RAMWR	0-F hex	word in ram
11,02	MMTORAM	BBBB in ALPHA flags 0, 1 and 3	stored program
11,03	AFAT	UOPAAA in ALPHA	FAT updated
11,04	DFAT	OPAAA in ALPHA	FAT updated
11,05	MOVE	BBBBEEEEDDDD in ALPHA	block is moved
11,06	CLBL	P / BBBBEEEE in ALPHA	block cleared
11,07	COPYR	SD in ALPHA	copied block
11,08	ROMSUM	P in ALPHA	romsum in xFFF
11,09	REG>ROM	0/P/BBBB in reg Y first reg in X	data in ram
11,10	---		
11,11	COMPILE	name of program	compiled program
11,12	LOCA	BBBBDDD in ALPHA	AAAADDD / NONE
11,13	LROM	BBBB in ALPHA	AAAADDD / NONE
11,14	COD	hex in ALPHA	binary in X
11,15	DECOD	binary in X	hex in ALPHA
11,16	ROMCHKX	XROM in X	bad / ok do if true
11,17	ROM>REG	BBBBEEEE in reg Y first reg in X	data in registers last reg in LASTX
11,18	MNEM	AAAADDD in Y	mnemonic in Z and T
11,19	DISASM	BBBB in X	BBBB + 1 in X AAAADDD in Y
11,20	CAT	P at prompt	cat from page P
11,21	CBT	name at prompt	length of program
11,22	SYNT	X first dec. byte Y second dec. byte	instruction after pc.
11,23	GE	pc. at .END.	

APPENDIX A

XROM	NAME	INPUT	OUTPUT
11,24	---		
11,25	SAVEROM	name in ALPHA dec. page in X	4K in file on tape
11,26	GETROM	name in ALPHA dec. page in X	4K of tape in ram
11,27	CMPDL	name of program	short comp. program
11,28	IPAGE	name in ALPHA dec. page in X xrom at prompt	desired page cleared name + xrom in page load addr. in ALPHA
11,29	MKPR	name of program	private program

SHORT FORM LETTER	REPRESENTING
A	address digit
B	begin address digit
D	data digit or destination digit
E	end-address digit
O	offset digit
P	page number digit
S	source digit
U	user digit

APPENDIX B

PROGRAMMING AND THE MLDL EPROM SET

Most functions provided by the **ERAMCO MLDL-EPROM** can be entered in program whenever the eprom-set is plugged in an **ERAMCO MLDL-box** connected to the calculator. When the **ERAMCO MLDL-box** containing the eprom set is connected program lines with eprom functions are displayed and printed as standard functions.

If the box is disconnected, these program lines are displayed and printed as XROM functions with two identification numbers. The first number -11- indicates that the functions are provided in the **ERAMCO MLDL-EPROM.** The second number identifies the particular function. The XROM numbers for the **ERAMCO MLDL-EPROM** are listed below.

Function	XROM Number	Function	XROM Number	Function	XROM Number
AFAT	XROM 11,03	DISASM	XROM 11,19	RAMWR	XROM 11,01
CAT	XROM 11,20	GE	XROM 11,23	REG>ROM	XROM 11,09
CBT	XROM 11,21	GETROM	XROM 11,26	ROMCHKX	XROM 11,16
CLBL	XROM 11,06	IPAGE	XROM 11,28	ROMSUM	XROM 11,08
CMPDL	XROM 11,27	LOCA	XROM 11,12	ROM>REG	XROM 11,17
COD	XROM 11,14	LROM	XROM 11,13	SAVEROM	XROM 11,25
COMPILE	XROM 11,11	MKPR	XROM 11,29	SYNT	XROM 11,22
COPYR	XROM 11,07	MNEM	XROM 11,18	---	XROM 11,10
DECOD	XROM 11,15	MMTORAM	XROM 11,02	---	XROM 11,24
DFAT	XROM 11,04	MOVE	XROM 11,05		

Underlined functions are not programmable.

If program lines using the **ERAMCO MLDL** eprom are entered when the eprom set is not connected, the function is recorded and displayed as XEQ followed by the function name. Program execution will be slowed down by lines in this form because the calculator will first search in main memory for a program or program line with the specified label.

APPENDIX C

MESSAGES

This is a list of messages and errors related to the functions in the **ERAMCO MLDL-EPROM set**. When any of these errors are generated the attempted function is not performed, except as noted.

DISPLAY	FUNCTION	MEANING
BAD MLDL	RAMWR	The **MLDL** ram page is malfunctioning.
ENTRY>64	AFAT	There are already 64 entry's in the FAT.
GTO/XEQ IND	CMPDL MMTORAM	The program contains GTO or XEQ ind statements.
NO ENTRY	DFAT	No such entry exists in the FAT.
NO HPIL	SAVEROM GETROM	The HPIL module is not plugged in.
NO LBL xx	COMPILE CMPDL MMTORAM	The GTO or XEQ has no corresponding LBL in this program.
NONE	LROM LOCA	The whole block is empty. There is no such word in the block from start-address up to the end of the page.
NONEXISTENT	-all- ROM>REG	The **ERAMCO MLDL-EPROM set** is not plugged in or is disabled or is malfunctioning. There are not enough registers available to store the specified block.
NO ROM	RAMWR	An attempt has been made to write to an page which does not have a valid XROM number at the first address of this page.
NO ROM xx	ROMCHKX	The ROM with the given XROM number is not plugged in or disabled.

APPENDIX C

DISPLAY	FUNCTION	MEANING
NO WRITE	RAMWR	The data is not written at the desired address. It is impossible to write to an EPROM or ROM page. Also you can not write at a disabled page.
PAGE > 15	GETROM IPAGE SAVEROM	There is an invalid pagenumber in reg X.
ROM	MKPR MMTORAM COMPILE CMPDL CBT	The named program doesn't exist in main memory but is found in ROM
xx NN-RR BAD	ROMCHKX	The ROM with the XROM number xx is bad.
xx NN-RR OK	ROMCHKX	The ROM with the XROM number xx is ok.
COMPL 2B G	COMPILE CMPDL MMTORAM	The 2 byte GTO's are handled.
COMPL 3B G/X	COMPILE CMPDL MMTORAM	The 3 byte GTO's and XEQ's are handled.
LOADING PGM	MMTORAM	The program is loaded to **MLDL** ram.
PACKING	COMPILE CMPDL MMTORAM	A byte is deleted and the program is packed to reduce the length of the program.
READY	COMPILE CMPDL IPAGE MMTORAM	The function is ready.

APPENDIX D

XROM numbers range from 1 up to 31 inclusive. As quite a few ROM's are available at the moment of this writing it is advisable to choose a XROM number with care to avoid conflicts with other modules.

ROM name	XROM ID	ROM name	XROM ID
MATH	01	SECUR	19 *
STAT	02	CLINLAB	19 *
SURVEY	03	AVIATION	19 *
FINANCE	04	MONITOR	19 * +
STANDARD	05	STRUCT-B	19 *
CIR ANAL	06	C PPC 1981	20
STRUCT-A	07	ASSEMBLER 3	21
STRESS	08	IL-DEVEL	22
HOME MN	09	I/O	23
GAMES	10 *	IL-DEVEL	24
C PPC 1981	10 *	-EXTFCN	25
AUTODUP	10 *	-TIME-	26
REAL EST	11	- WAND	27
MACHINE	12	-MASS ST	28
THRML	13	(- CTL FNS -	
NAVIG	14	HP-IL MODULE)	
PETROL	15	-PRINTER	29
PETROL	16	CARD READER	30
PLOTTER	17	PPC ROM 2 ??	31
PLOTTER	18	ERAMCO-**MLDL**	11

+ Only a small number of this ROM, an early version of IL-DEVEL ROM, were made and are not stocked or sold by HP.

Those marked with an asterisks share their identifying number, and should not be used in the HP-41 at the same time. Of two functions with the same XROM ID the one at the lowest address (i.e. the lowest numbered port) will be accessed first and the other will be ignored. So use discretion when choosing your own XROM number if you want to avoid these kind of problems.

APPENDIX E

X R O M S T R U C T U R E

XROM's are located at whole 4k blocks of addresses. The lowest addresses in an XROM, and a few of the highest have special functions. The remainder may be filled in any way. The locations in the 4k blocks must be filled by ten bit words, giving 2^10 different codes. They may be read as instructions, or as alpha-numeric data. The following summary, adapted from J. Schwartz' January 1983 PPC Conference paper, should be taken into account when studying an application ROM, e.g. the **MLDL**-ROM. A listing can easily be prepared by using the **MLDL**-ROM functions DISASM and MNEM.

Relative address (hex)	Function of code at that address
X000	The XROM ID number in hexadecimal digits.
X001	The number of functions in the XROM (m), including the XROM name.
X002-3	Address of XROM name
X004-5	Address of first routine, program, etc.
X005-7	Address of second routine, etc.
"	" "
"	" "
X002+2n	Address of n'th routine
X003+2n	
"	" "
"	" "
X002+2m	Address of last (m'th) routine
X003+2m	(m < 64)
X004+2m	Compulsory null - 000.
X005+2m	Compulsory null - 000.
"	" "

```
Add. of name   Name of ROM (running backwards)
   "                    "            "
   "                    "            "
Add. of Fn# 1  Start of Fn# 1 code
   "                    "            "
   "                    "            "
Add. of Fn# 2  Start of Fn# 2 code
   "                    "            "
   "                    "            "
   "                    "            "
 XFF4-A        Special interrupt jump locations ( see table ).
 XFFB-E        ROM name abbreviation and revision #.
 XFFF          ROM checksum for diagnostic use
```

Word pairs containing function addresses:

First word of pair:	b	0	0	0	0	0	a11	a10	a9	a8
Second word of pair:	0	0	a7	a6	a5	a4	a3	a2	a1	a0

This results in the following address in this 4k block if 0000 is zero:

p3 p2 p1 p0 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0

Where p0-3 is the bit representation of the 4k page number and a0-11 represent the relative offset from the beginning of the page.When 0000 is not equal to zero it must be added to p0-3. For more information see the function AFAT.

If the two words would read 003, 0FF this would represent a starting address of a function at address X3FF (hex). The bit b in the first word indicates USER code or microcode. If set the address is the start of a USER code program (e.g. 200, 0A1 in the printer module is address 60A1, start of USER code program "PRPLOT")

APPENDIX F

THE SPECIAL INTERRUPT POINTS

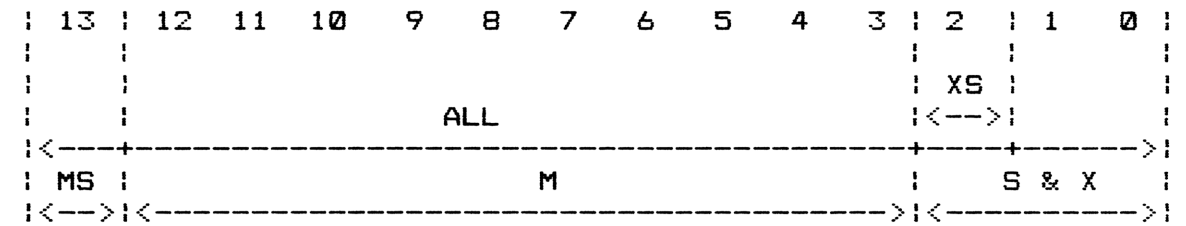
xFF4 Interrupts during PSE loop.
xFF5 Interrupts after each program line.
xFF6 Wake-up with no key down.
xFF7 Interrupts when turned off.
xFF8 Interrupts when peripheral flag is set.
xFF9 wake-up with ON key.
xFFA Wake-up after memory lost.

Do not use these points unless you know exactly what you are doing. Careless use of these points may cause CRASHES.

ASSEMBLY LANGUAGE INFORMATION

SHORT REVIEW OF THE HP-41 INSTRUCTIONS

The HP41 CPU has three main arithmetic registers: A,B and C. These are 56 bits long (14 nibbles) and instructions can operate in various "fields" of the register.



ALL : The whole register
M : Mantissa
MS : Mantissa Sign
XS : eXponent Sign
S&X : eXponent and Sign off exponent

@R : At specified pointer
R<- : from digit R to digit 0
PQ : Between P and Q

There are two pointers P and Q, of which the value is 0-13. One of them is selected at the time (through slct p or slct q), the selected pointer is called R. These are three extra fields, which depend on the value of the pointer), R<- (up to R, from digit R to digit 0) and P-Q (between pointer P and Q, Q must be greater than P).

There is a register G, 8 bits long, that may be copied to or from or exchanged with the nibbles R and R+1 of register C. (R<=12). There are 14 flags, 0-13, of which flags 0-7 are located in the 8-bits ST (status) register, and there is a 8-bits TONE register T, of which the contents floats every machine cycle through a speaker.

Then there are two auxilary storage registers, M and N, which can operate only in the field ALL. They are 56 bits long.

There is a 16-bit program counter, which addresses the machine language, and a KEY register of 8-bits, which is loaded when a key is pressed. The returnstack is 4 addresses long and is situated in the CPU itself.

The CPU may be in HEX or DEC mode. In the last mode the nibbles act as if they can have a value from 0 to 9.

The USER-code RAM is selected by C[s&x] through RAM SLCT, and can be written or read through WRITE DATA or READ DATA. If chip 0 is selected (RAM address 000 to 00F) the 16 stack registers may be addressed by WRIT and READ 0 to 15.

Peripherals (such as display, card reader, printer) may be selected by C[s&x] through PRPH select or by SELP (see page 19).

The mnemonics are a kind of BASIC structure.

Arithmetic instructions (operate on a specified field)

A=0	C=B	C=C+1	?A<B
B=0	A=A+1	C=C+A	?A#C
C=0	A=A+B	C=A-C	?A#0
A<>B	A=A+C	C=0-C	RSHFA
B=A	A=A-1	C=-C-1	RSHFB
A<>C	A=A-B	?B#0	RSHFC
A=C	A=A-C	?C#0	LSHFA
C<>B	C=C+C	?A<C	

CLRF, SETF, ?FSET, ?R=. ?FI (peripheral flag set?) , RCR (rotate right) have a parameter 0-13.

LD@R (load C at R) and SELP (select peripheral) have a parameter 0-F.

WRIT and READ have a parameter 0-15, called
0(T), 1(Z), 2(Y), 3(X), 4(L), 5(M), 6(N), 7(O), 8(P), 9(Q),
10(!-), 11(a), 12(b), 13(c), 14(d), 15(e).

Jumps:
There are two classes jumps:

a. JNC (jump if no carry) and JC (jump if carry). These instructions provide to jump relative 3F in positive direction or 40 in negative direction.

b. ?NC GO and ?C GO. These instuctions provide to jump to an absolute 16 bits address.

?NC XQ and ?C XQ are jump-subroutine instructions to absolute addresses. (remember the return stack is just 4 addresses long).

Miscelaneous instructions:

ST=0	C=G	ST=T	POWOFF
CLRKEY	C<>G	ST<>T	SLCT P
?KEY	C=M	ST=C	SLCT Q
R=R-1	M=C	C=ST	?P=Q
R=R+1	C<>M	ST<>C	?LOWBAT
G=C	T=ST	XQ->GO	A=B=C=0
GOTO ADR (C[6:3])	?C RTN	PUSH (C[6:3])	
C=KEY	?NC RTN	POP (C[6:3])	
SETHEX	RTN	GOTO KEY	
SETDEC	N=C	RAM SLCT	
DSPOFF	C=N	WRITE DATA	
DSTTOG	C<>N	READ DATA	
FETCH S&X	C=C or A	PRPH SLCT	
WRIT S&X (for MLDL)	C=C and A		

Note : various arithmetic and all test instuctions may set the carry flag. This flag keeps set only one machine cycle, so a jump dependent on this flag must be immediate after the arithmetic or test instruction, otherwise the carryflag will always be cleared.

CLASS 0 OPERATIONS

p=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NOP	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
CLRF p	384	304	284	004	044	084	144	284	104	244	0C4	184	344	2C4	---	---
SETF p	388	308	288	008	048	088	148	288	108	248	0C8	188	348	2C8	---	---
?FSET p	38C	30C	28C	00C	04C	08C	14C	28C	10C	24C	0CC	18C	34C	2CC	---	---
LD@R p	010	050	090	0D0	110	150	190	1D0	210	250	290	2D0	310	350	390	3D0
?R= p	394	314	214	014	054	094	154	294	114	254	0D4	194	354	2D4	---	---
R= p	39C	31C	21C	01C	05C	09C	15C	29C	11C	25C	0DC	19C	35C	2DC	---	---
SELP p	3A4	324	224	024	064	0A4	164	2A4	124	264	0E4	1A4	364	2E4	1E4	3E4
WRIT p	028	068	0A8	0E8	128	168	1A8	1E8	228	268	2A8	2E8	328	368	3A8	3E8
?FI p	3AC	32C	22C	02C	06C	0AC	16C	2AC	12C	26C	0EC	1AC	36C	2EC	---	---
READ p	038	078	0B8	0F8	138	178	1B8	1F8	238	278	2B8	2F8	338	378	3B8	3F8
RCR p	3BC	33C	23C	03C	07C	0BC	17C	2BC	13C	27C	0FC	1BC	37C	2FC	---	---

MNEMONIC		OPERATION
NOP		No operation
CLRF	p	Clears system flag number p
SETF	p	Sets system flag number p
?FSET	p	Set the carry flag if system flag p is set
LD@R	p	Load p into "C" at nibble pointed at by pointer and decrement pointer
?R=	p	Set the carry flag if the active pointer equals p
R=	p	Set the active pointer to p
SELP	p	Transfer control to the desired peripheral p
WRIT	p	Write "C" to RAM memory or to the selected device in register p of the selected block
?FI	p	Set the carry flag if peripheral flag p is set
READ	p	Read "C" from RAM memory or the selected device to register p in the selected block
RCR	p	Rotate "C" right by p digits

CLASS 0 SPECIAL INSTRUCTION HEX CODES

MNEMONIC	HEX	OPERATION
UNUSED	x34	Not in use
UNUSED	x74	" "
UNUSED	xB4	" "
UNUSED	xF4	" "
ST=0	3C4	Clears flag 0 to 7 ("ST" register)
CLRKEY	3C8	Clears the 'key pressed' flag
?KEY	3CC	Set the carry flag when a key has been pressed
R=R-1	3D4	Decrement the current pointer
R=R+1	3DC	Increment the current pointer
UNUSED	018	Not in use
G=C	058	Copy digits *r,r+1* from "C" to "G"
C=G	098	Copy "G" into digits *r,r+1* from "C"
C<>G	0D8	Exchange "G" with digits *r,r+1* from "C"
UNUSED	118	Not in use
M=C	158	Copy "C" into "M"
C=M	198	Copy "M" into "C"
C<>M	1D8	Exchange "C" with "M"
UNUSED	218	Not in use
T=ST	258	Copy "ST" into "T"
ST=T	298	Copy "T" into "ST"
ST<>T	2D8	Exchange "ST" with "T"
UNUSED	318	Not in use
ST=C	358	Copy digits 1, 0 from "C" into "ST"
C=ST	398	Copy "ST" into digits 1, 0 from "C"
C<>ST	3D8	Exchange digits 1, 0 from "C" with "ST"
XQ->GO	020	Drop stack to convert XQ into GO
POWOFF	060	Go to standby mode
SLCT P	0A0	Select "P" as the active pointer
SLCT Q	0E0	Select "Q" as the active pointer

MNEMONIC	HEX	OPERATION
C=KEY	220	Copy key register into digit 4, 3 of "C"
SETHEX	260	Use hexadecimal arithmetic
SETDEC	2A0	Use decimal arithmetic
DSPOFF	2E0	Turn off the display
DSPTOG	320	Toggle the state of the display
C RTN	360	Return from subroutine if the carry is set
NC RTN	3A0	Return from subroutine if carry flag clear
RTN	3E0	Do a subroutine return always
UNUSED	030	Not in use
N=C	070	Copy "C" into "N"
C=N	0B0	Copy "N" into "C"
C<>N	0F0	Exchange "C" with "N"
LDI	130	Load next rom word into digits 2-0 of "C"
PUSH	170	Push address digits 6-3 in "C" onto stack
POP	1B0	Pop address from stack into digits 6-3 of
UNUSED	1F0	Not in use
GOTO KEY	230	Load key register into lower 8 bits of "PC
RAM SLCT	270	Set ram address to digits 2-0 of "C"
UNUSED	2B0	Not in use
WRITEDATA	2F0	Write register "C" to the selected registe
FETCH	330	Load 2-0 of "C" from rom address 6-3 of "C
C=C OR A	370	Logical *or* of "C" with "A" bit by bit
C=C AND A	3B0	Logical *and* of "C" with "A" bit by bit
PRPHSLCT	3F0	Set peripheral address to digit 2-0 of "C"
?P=Q	120	Set the carry flag if the pointers are equ
?LOWBAT	160	Set the carry flag if low battery
A=B=C=0	1A0	Clear registers "A" "B" and "C"
GOTO ADR	1E0	Copy digits 6-3 of "C" into the "PC"

CLASS 1 INSTRUCTIONS

Class 1 instructions are absolute GOTOs and EXECUTEs. They consist of two consecutive ROM words of the following format :

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	0	1
A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	*p*	*p*

A_{15}-A_0 is the 16-bit address to branch to. The *pp* field of the second word determines what type of instruction it is. The next table shows values for *pp* :

pp	MNEMONIC	OPERATION
00	NC XQ	execute subroutine if carry is clear
01	C XQ	execute subroutine if carry is set
10	NC GO	goto rom address if carry is clear
01	C GO	goto rom address if carry is set

Example : NC GO 0232 which jumps to the memory lost routine is coded as :

0011 0010 01 = 0C9 as first word
0000 0010 10 = 00A as second word

CLASS 2 FIELDS OF OPERATION

FIELD	AREA OF OPERATION
ALL	All digits.
M	Mantissa digits 12 - 3.
MS	Mantissa sign digit 13.
XS	Exponent sign digit 2.
S&X	At exponent digits 2 - 0.
@R	At digit specified by the current pointer.
R<-	Up to and including pointer from the right.
PQ	from pointer P, left up to Q, including pointers.

CLASS 2 INSTRUCTIONS

MNEMONIC	OPERATION	@R	S&X	R<-	ALL	PQ	XS	M	S
A=0	clear A	002	006	00A	00E	012	016	01A	01E
B=0	clear B	022	026	02A	02E	032	036	03A	03E
C=0	clear C	042	046	04A	04E	052	056	05A	05E
A<>B	exchange A with B	062	066	06A	06E	072	076	07A	07E
B=A	copy A into B	082	086	08A	08E	092	096	09A	09E
A<>C	exchange A with C	0A2	0A6	0AA	0AE	0B2	0B6	0BA	0BE
C=B	copy B into C	0C2	0C6	0CA	0CE	0D2	0D6	0DA	0DE
C<>B	exchange B with C	0E2	0E6	0EA	0EE	0F2	0F6	0FA	0FE
A=C	copy C into A	102	106	10A	10E	112	116	11A	11E
A=A+B	add B into A	122	126	12A	12E	132	136	13A	13E
A=A+C	add C into A	142	146	14A	14E	152	156	15A	15E
A=A+1	increment A	162	166	16A	16E	172	176	17A	17E
A=A-B	subtract B from A	182	186	18A	18E	192	196	19A	19E
A=A-1	decrement A	1A2	1A6	1AA	1AE	1B2	1B6	1BA	1BE
A=A-C	subtract C from A	1C2	1C6	1CA	1CE	1D2	1D6	1DA	1DE
C=C+C	double C	1E2	1E6	1EA	1EE	1F2	1F6	1FA	1FE
C=A+C	add A into C	202	206	20A	20E	212	216	21A	21E
C=C+1	increment C	222	226	22A	22E	232	236	23A	23E
C=A-C	A-C into C	242	246	24A	24E	252	256	25A	25E
C=C-1	decrement C	262	266	26A	26E	272	276	27A	27E
C=0-C	complement C	282	286	28A	28E	292	296	29A	29E
C=-C-1	nines complement C	2A2	2A6	2AA	2AE	2B2	2B6	2BA	2BE
?B≠0	set carry flag if B≠0	2C2	2C6	2CA	2CE	2D2	2D6	2DA	2DE
?C≠0	set carry flag if C≠0	2E2	2E6	2EA	2EE	2F2	2F6	2FA	2FE
?A<C	set carry flag if A<C	302	306	30A	30E	312	316	31A	31E
?A<B	set carry flag if A<B	322	326	32A	32E	332	336	33A	33E
?A≠0	set carry flag if A≠0	342	346	34A	34E	352	356	35A	35E
?A≠C	set carry flag if A≠C	362	366	36A	36E	372	376	37A	37E
RSHFA	shift A right 1 digit	382	386	38A	38E	392	396	39A	39E
RSHFB	shift B right 1 digit	3A2	3A6	3AA	3AE	3B2	3B6	3BA	3BE
RSHFC	shift C right 1 digit	3C2	3C6	3CA	3CE	3D2	3D6	3DA	3DE
LSHFA	shift A left 1 digit	3E2	3E6	3EA	3EE	3F2	3F6	3FA	3FE

CLASS 3 INSTRUCTIONS

DISTANCE	JNC-	JC-	JNC+	JC+	DISTANCE	JNC-	JC-	JNC+	JC+
+/- 01	3FB	3FF	00B	00F	+/- 02	3F3	3F7	013	017
+/- 03	3EB	3EF	01B	01F	+/- 04	3E3	3E7	023	027
+/- 05	3DB	3DF	02B	02F	+/- 06	3D3	3D7	033	037
+/- 07	3CB	3CF	03B	03F	+/- 08	3C3	3C7	043	047
+/- 09	3BB	3BF	04B	04F	+/- 0A	3B3	3B7	053	057
+/- 0B	3AB	3AF	05B	05F	+/- 0C	3A3	3A7	063	067
+/- 0D	39B	39F	06B	06F	+/- 0E	393	397	073	077
+/- 0F	38B	38F	07B	07F	+/- 10	383	387	083	087
+/- 11	37B	37F	08B	08F	+/- 12	373	377	093	097
+/- 13	36B	36F	09B	09F	+/- 14	363	367	0A3	0A7
+/- 15	35B	35F	0AB	0AF	+/- 16	353	357	0B3	0B7
+/- 17	34B	34F	0BB	0BF	+/- 18	343	347	0C3	0C7
+/- 19	33B	33F	0CB	0CF	+/- 1A	333	337	0D3	0D7
+/- 1B	32B	32F	0DB	0DF	+/- 1C	323	327	0E3	0E7
+/- 1D	31B	31F	0EB	0EF	+/- 1E	313	317	0F3	0F7
+/- 1F	30B	30F	0FB	0FF	+/- 20	303	307	103	107
+/- 21	2FB	2FF	10B	10F	+/- 22	2F3	2F7	113	117
+/- 23	2EB	2EF	11B	11F	+/- 24	2E3	2E7	123	127
+/- 25	2DB	2DF	12B	12F	+/- 26	2D3	2D7	133	137
+/- 27	2CB	2CF	13B	13F	+/- 28	2C3	2C7	143	147
+/- 29	2BB	2BF	14B	14F	+/- 2A	2B3	2B7	153	157
+/- 2B	2AB	2AF	15B	15F	+/- 2C	2A3	2A7	163	167
+/- 2D	29B	29F	16B	16F	+/- 2E	293	297	173	177
+/- 2F	28B	28F	17B	17F	+/- 30	283	287	183	187
+/- 31	27B	27F	18B	18F	+/- 32	273	277	193	197
+/- 33	26B	26F	19B	19F	+/- 34	263	267	1A3	1A7
+/- 35	25B	25F	1AB	1AF	+/- 36	253	257	1B3	1B7
+/- 37	24B	24F	1BB	1BF	+/- 38	243	247	1C3	1C7
+/- 39	23B	23F	1CB	1CF	+/- 3A	233	237	1D3	1D7
+/- 3B	22B	22F	1DB	1DF	+/- 3C	223	227	1E3	1E7
+/- 3D	21B	21F	1EB	1EF	+/- 3E	213	217	1F3	1F7
+/- 3F	20B	20F	1FB	1FF	+/- 40	203	207	---	---

Class 3 instructions allow the program to jump up to 63 words forward or backward from its present location. The mnemonics are JNC and JC.

ROM CHARACTER TABLE

lower 4 / upper 2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
4	⊢	a	b	c	d	e								≠	

Note : The colon (3A) displays as a boxed star. The comma (2C) is also the left facing goose when used in a function name or display and the period (2E) is also the right facing goose.

You get the hexadecimal code of a character by taking the number in the upper2 column and place the number in the lower row behind it. Last step is to place a zero in front of the number.

Example : The hexadecimal code of the letter W is 017.
Of the equal sign it is 03D

FUNCTION NAMES

When a function is executed, the operating system checks the ROM words containing the first two characters of the function name and the two words immediately following. The catalog table entry for a microcode function (both mainframe and XROM functions) points to the first word of executable code. The function name is listed in reverse order immediately preceding the first word of executable code.

Example : This example shows you how a normal function name is coded.

```
10CE  081  A   Hex 080 added to indicate end of name.
10CF  00C  L
10D0  003  C
10D1  xxx      First executable word of CLA.
```

FUNCTION PROMPTING

To tell the operating system that the end of the function name has been reached, add 080 hex to the final character. To provide a prompt set the top two bits in the first two characters of the function name by adding the hex constants in the following table :

1ST	2ND	NULL alpha	alpha	#dig.	IND & ind stack	stack	none	example
000	any						X	CLA,CLST
100	000	X	X					CLP,COPY
100	100			3,4				SIZE
100	200		X					
100	300			1	X			CAT,TONE
200	000			2	X	X		STO,RCL
200	100			2	X	X		STO,RCL
200	200			2	X			FS?,SF
200	300		X	2	X			
300	000		X	2				LBL
300	100		X	2	X			XEQ(alpha)
300	200		X	2				
300	300		X	2	X X(.ddd)			GTO

The operating system examine these ROM bits and executes a prompt (if the appropiate bits are set) before the function is executed. These prompts are only executed when you execute the function from the keyboard. However, when the function is executed in a program there will be no prompt at all. **Take care of this.**
If the prompt accepts an alpha string, the input data is loaded into the Q register, right justified in reverse order in ASCII.

Example : Execution of the function ASN with the alpha argument "COPY" will load 00 00 00 59 50 4F 4C into the Q register before the function is executed.

If the prompt is numeric the input data is loaded into the "A" register in binary. Whenever the prompt also accepts indirect, the value in the "A" register is increased with hex 60.

Example : Execution of the function RCL with a numeric argument of 55 will return 00 00 00 00 00 00 37 in the "A" register.
If the prompt would have been filled in with IND 55, the "A" register contains 00 00 00 00 00 00 B7.

PROGRAMMABILITY

Two other ROM words of a microcode function are examined by the operating system. The first executable word, if a nop (000), indicates that the function is non-programmable. This means that if you execute the function in program mode, it executes rather than being entered as a program line. SIZE, ASN and CLP are non-programmable functions.
If the first two executable words of a XROM function are both zero, then the function is both non-programmable and executes immediately. This means that no function name is displayed and that the function will not NULL. The function is executed when the key is pressed rather than when the key is released. PRGM, SHIFT and back-arrow are non-programmable, immediate executing functions. Note that unless your routine checks for key release, and the key to which your function is assigned is held down, the function will be executed repeatedly until the key is released. These two words affect the function operation only if the calculator is in PRGM mode. In RUN mode, they are ignored.

Example : these are a few examples of function name promptings.

12D2	097	W	1105	099	Y	12CC	085	E
12D3	005	E	1106	010	P	12CD	00E	N
12D4	109	I	1107	00F	O	12CE	30F	O
12D5	216	V	1108	103	C	12CF	114	T

FUNCTION INDEX

CARE AND WARRANTY

Eprom care

Store the eprom set in a dry and clean place. Make sure that the feet of the eprom's are protected against bending. Otherwise a pin could brake from the eprom and make it worthless. Do not connect any external power supply to the eproms. Protect the eproms against static charges, otherwise irrepairable damage to the eproms can result. **Do not** remove under any circumstances the labels on the eproms for these labels protect the eproms against loosing there data by accident through too much U.V. light on the eprom's.

Limited 180 day's warranty

The 83120A **ERAMCO MLDL-Eprom set** is warranted against defects in materials and workmanship affecting electronic performance, -but not software content- for 180 day's from the date of original purchase. If you sell your unit or give it as a gift the warranty is automatically transferred to the new owner and remains in effect for the original 180 days period. During the warranty period we will repair or at our option replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to **ERAMCO SYSTEMS** or their official service representative.

CARE AND WARRANTY

WHAT IS NOT COVERED

This warranty doesn't apply if the product has been damaged by accident, misuse or as the result of service or modification by other than **ERAMCO SYSTEMS** or their official service representative.

No other express warranty is given. Any other implied warranty of merchantability or fitness is limited to the 180 days period of this written warranty. In no event shall **ERAMCO SYSTEMS** be liable for consequential damages. This liability shall in no way exceed the catalog price of the product at the moment of sale.

Obligation to Make Changes

Products are sold on the basis of specifications applicable at the time of manufacture. **ERAMCO SYSTEMS** shall have no obligation to modify or update products once sold.

HOW TO SET UP YOUR OWN EROM PAGE

This part of the manual wiil tell you exactly how to set up an Erom image in your **MLDL**-box. This is done with the help of a few user code routines that are loaded into the **MLDL** Erom pages. If you follow the instructions to the letter, nothing can go wrong. And with the help of these instructions you should be able to set up your own Erom image.

step 1

The first thing that has to be done is to clear the Erom page you want to work at and to set the Erom block to the proper page. Therefore you must set the first block with the left rotary switch at page A. Set the rotary switch of the other block to page E. Disable both the switches to the left of the leftmost rotary switch (pull them down). When you set the switches in this position, you can compare the results of your actions with the results that will be given in this appendix.

step 2

Now we will first clear both Erom pages. Key in alpha mode the single character "A". Go out alpha mode and execute CLBL (for more details see page 14) Repeat this sequence with the single character "E" in alpha. At this moment your Erom pages should both be clear. Now you can enable both the Erom pages by pushing the both switches up. Don't expect anything to happen yet. Both pages are still empty.

step 3

Before doing anything else we have to make sure that both pages are empty. Key in alpha "AFFF". Now execute LROM. The display should read 'none'. If this is not the case you should control the setting of the switches and try step 2 again. This is done in the same way for the second block, except you now have to key in alpha "EFFF". The reading of the display should be again 'none'. If this isn't the case return to step 2.

step 4

To allow the HP-41 to find anything that is plugged into the system it uses the first word on every page starting from page 5. If this word doesn't contain a valid identifier, it can't execute a routine or function located at that page. Therefore we will continue with the setting of these identifiers for both Erom pages. In fact this identifier is the xrom number of a module. To avoid any problems with other modules it is recommended in this stage to unplug all your modules.

Also the name of the rom module has to be added. For this the function IPAGE is used. It is enough to put the rom name into the ALPHA register. After this you give the 4K page address in the X register. Now you can execute the function IPAGE. It will prompt you for a XROM number. To avoid problems we choose as XROM number the number 21.

Note : In this manual we described two ways to set up an Erom image. First time we did this with the function RAMWR (see page 5). For this is quite a cumbersome way to prepare an Erom image we did incorporate the function IPAGE (see page 35). Here we already gave you an example of how to create your own Erom image.

Example : We will create one Erom image with xrom number 21 and as name "TEST ROM 1A". For this we make use of the RAM page that is controlled by the left rotary and enabling switch. The block is already cleared and enabled in step 2. The block is addressed at page "A". Now we have all relevant data for the block, so we can initialize it.
Key into ALPHA the name of the module and into the X register the address of the RAM page that will hold the Erom image. This address is 10.
Execute the function IPAGE. At the prompt you answer with the desired xrom number E.G. 21. After a while a tone will sound and the message READY is displayed.

step 5

From now on the HP-41 can recognize anything that is written into Erom block one. So lets give it a try. First of all we have to create a little program in main memory that is to be stored in the Erom block.

We will use the following program:

```
LBL 'test
LBL 01
BEEP
GTO 01
END
```

step 6

You have now created a program in the memory of your calculator. But we wanted to have this program in the **MLDL**-box, because it is using up the last free bytes we had. That's no problem. We only have to use MMTORAM to get the program in the Erom page we want it. For this we have to initialize a few things.

When we have initialized our Erom page manually (without use of IPAGE), we have to give the starting address for our program. This address will be the first word to be used by MMTORAM. Do not use the reserved words in an Erom image in which you are to load your programs (see appendix E and appendix F).
If you work with IPAGE however, the starting address is already given in the ALPHA register. When you have to use the ALPHA register between two sessions of loading programs, it is advisable to keep the contents of the ALPHA register in a normal data storage register, or to note it down (be carefull saving the address in a storage register, for MMTORAM can clear all the user registers, when it makes use of CMPDL). This is handy for future use. If you lost the address however, you can find it back with the help of LROM. Increase the address given by LROM with one, and you have the new starting address to store at.

Second thing we have to initialize is the setting of flags 0 and 1, to achieve the desired private status of the loaded program. There are four options for these flags. For a full description of these options we rever to the function MMTORAM at page 8.

Third and last initialisation we have to make is the setting of flag 3. MMTORAM decides on this flag wether it shall use CMPDL or the normal COMPILE function when it is loading a program. See the function CMPDL for the difference between the two compilers.

Example : We are going to load the program described at step 5. This program has to be loaded in a nonprivate, complete open status. Furthermore we do not want the numeric labels to be deleted.
We do not have to give the starting address, for this is given in ALPHA by the function IPAGE.
For a complete open, nonprivate status flags 0 and 1 have to be cleared.
Flag 3 has to be set for we do not want the numeric labels to be deleted.

When these settings are made, the function MMTORAM can be executed. You will see the messages of the compiler and then the message "LOADING PGM". When MMTORAM is finished a tone will sound and the message "READY" is displayed. The program is now loaded in the Erom image and is ready for use.

Note : If you switch to ALPHA you will see that the starting address is changed. It now points to the first free byte after the just loaded program. This provides an easy way of loading subsequent programs.

step 7

First thing we will do is deleting the program from main memory. When you have done this, you should still be able to execute test for it has been stored in the Erom page. So give it a try. You will hear the familiar beeping every time the program is looping. Stop execution of the program and switch to PRGM mode. Whenever you try to insert or delete a program step, you will see the message 'ROM'. This proves that the program has realy been loaded into the **MLDL**-box. The program is also included in catalog 2. If you execute CAT 2 you will see the label test showing up in your display sooner or later, depending on the amount of other roms that are plugged into the system.

When you want to store more and other programs, you can follow the described procedure starting at step 5.

Load also the programs described on page 21 (TST) and 28 (MDIS). Load the TST program with flag 3 cleared. Look at the program after you have deleted it in main memory. As you will see, it does not contains the numeric labels any more. This and the fact that it is in ROM now, will speed up the execution quite a lot. Load the MDIS program with flags 1 and 3 set. The program will be open in the erom page, but as soon as it is copied back to main memory, it will be private.

This is the end of the description of our MLDL ROM operating system. We hope you will enjoy to work with this rom. If you have any complaints or wishes you want to see in a future rom, please let us know. We will take these in account as much as possible.

ERAMCO SYSTEMS
W. van Alcmade str. 54
1785 LS Den Helder
The Netherlands