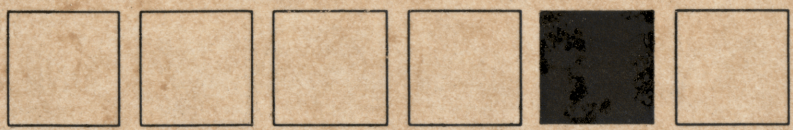
W&W SOFTWARE PRODUCTS

RAMBOX

Owner's Manual

For the HP-41



CONTENTS

4 FILE FUNCTIONS

1 INTRODUCTION

The following chapters 1.3 and 1.4 will give you an insight into the ROM-structure of your HP-41 and the construction of I/O buffers. Chapter 1.1 will give you help in installing the RAMBOX and tell you about its manifold possibilities. Chapter 1.2 is meant to explain the general working manner when using the RAMBOX.

1.1 INSTALLATION

IMPORTANT INSTRUCTIONS:
Make sure that your calculator is turned OFF when plugging in or removing the RAMBOX; otherwise it is possible that either your calculator or the RAMBOX will be damaged.

Before the very first operating of the RAMBOX remove all plug-in modules including the X-FUNCTION-Module (this is not necessary when using an HP-41 CX), excepting the QUAD-RAM, TIME- and X-MEMORY-Module.

The RAMBOX that you have acquired grasps a memory area of 32kB. 4 kB of this are reserved for the built-in operating system which puts 32 new functions at your disposal. These functions make a comfortable management of the entire RAMBOX memory area possible.

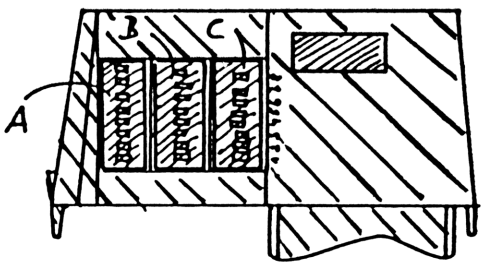
After you have plugged the RAMBOX into your calculator please turn on the calculator and feed it the following program:

```
01 *LBL'LP
02  8.015
03 *LBL 01
04  SF 25
05  LDPGM       (Function of the RAMBOX operating system)
06  FS?C 25
07  GTO 02
08  ISG X
09  GTO 01
10 'NO
11  PROMPT
12  LBL 02
13 'FINISHED
14  PROMPT
15  END
```

To load all the programs contained in your memory into the memory of the RAMBOX, just feed ALPHA the corresponding program name and execute the program "LP". If the program is being stored correctly, the messages "PACKING", "LOADING", "COMPILING", and "FINISHED" should show in that order. To make sure that your programs are being loaded you can execute CAT 2. Once you have stored all your programs in the RAMBOX in this manner, you may erase all programs contained in the calculator memory. The programs contained in the RAMBOX will be preserved! They can be immediately executed using XEQ "Program name" and can be put back into the calculator at any time, using the COPY-function.

When the calculator is turned off, you can remove the RAMBOX from it without a memory loss. The lithium battery which is built into the calculator saves your data from memory loss for 5 years if the RAMBOX is not plugged in. As long as the RAMBOX is on your calculator it is provided with electricity and will keep your data indefinitely, provided that the batteries are always in working order! Should the battery need to be exchanged, please send your RAMBOX to a W&W Service Center, which will do this work for the currently valid service price (half of the current hour rate plus the cost of the batteries).

The RAMBOX grasps a memory of 32k Byte, which is divided into eight 4k Byte blocks (from now on called pages). These eight blocks can be manipulated in manifold ways and manners by using the three groups of switches annexed to the underside of the RAMBOX.



The individual groups of switches have the following meaning:

Using the switches A1-A4 and the switches A5-A8, the first two 4kB blocks can be put on any 4kB area within the addressable 64kB. For this the address will have to be binarily coded using the switches. The pages 10-15 are permanently set and can not be altered!

The following rules are valid:

Address	Group A A1 A2 A3 A4 A5 A6 A7 A8
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Example: You want to put the first page on the address block 7, and the second page on the address block E. For this the switches have to be set as follows:

1. A2, A3, A4 on ON - A1 on OFF; this puts page 1 on address 7.
2. A5, A6, A7 on ON - A8 on OFF; now the second page is on address block E. The permanently addressed page E will now be cross faded (dissolved) and can not be used!

The group B makes it possible to protect each single page against writing over it and therefore also against accidental erasing of the page.

With the aid of group C every 4kB block can be turned off completely.

The switches B1, B2, and C1, C2 correspond with the pages whose addresses were set with the switches A1-A4, resp. A5-A8.

Example: Supposing that you want to put the two choice free pages on the blocks 8 and 9 and supply them with a write protection, turn off the pages A and B completely and use C,D,E, and F freely - without write protection. In this case the switch position will look as follows:

Switch group A -	A1, A5, and A8 are turned ON, all other switches are turned OFF (to the right).
Switch group B -	B1 and B2 on OFF, B3-B8 on ON
Switch group C -	C3 and C4 on OFF, all other switches of this block on ON.

1.2 GENERAL INSTRUCTIONS

Once you have set the first two pages of your RAMBOX on the addresses 8 and 9 (usually this is already set when the RAMBOX is delivered), you have 7 4kB blocks at your disposal for free programming in your RAMBOX. These blocks have the addresses 9,10,12,13,14, and 15. The operating system of your RAMBOX is on the address block 8. With its new functions this system allows for the use of your RAMBOX. Using this RAMBOX you can simulate 7 4k-modules (for example the mathematics module, statistics module, or similar). Contrary to the purchasable modules which can not be altered, the contents of the RAMBOX can be changed any time you like.

1.2.1 Page Titles

In order to be better able to find your programs in CATalog 2, you should give every initialized 4k block a "title". This "module name" is a simple name consisting of a maximum of 11 letters, and characterizes your module. Ordinarily every module name starts with a "-"-sign - in order to differentiate it from the other function names in a CATalog listing. The RAMBOX is pre-initialized with the module name -FREE PG 9-15 (see also INITPG). Following are several examples for page titles:

-MICHAELROM	<--	General title
-MATRIX WW1		
-SURVEY WW1	<--	Titles for the specification of groups of functions belonging together. Initials and the revision number can also be mentioned in the module name.

1.2.2 XROM Numbers

In order to be able to define its plug-in modules, the HP-41 uses 32 different numbers. Up to 64 different program names, resp. functions, can be managed by the calculator, always in connection with one of these numbers (per module, resp. 4k block). The number combinations are called "XROM-numbers", since the HP-41 shows the word "XROM", if you are looking at a program line or a key sequence when the corresponding module is not plugged in.

Since several modules can not be used with the same XROM number on the calculator, it is necessary for you to choose XROM numbers for your own 4k blocks, which your system does not otherwise need. Following you will find a list of the XROM numbers currently being used, so that you can avoid possible conflicts right from the beginning.

01 MATH
02 STATISTICS, DAVID ASSM
03 SURVEYING
04 FINANCIAL
05 STANDART, ZENROM
06 CIRCUITS, ADVANCED FUNCTIONS
07 STRUCTURAL ANALYSIS
08 STRESS
09 HOME MANAGEMENT, CCD MODULE
10 GAMES, AUTO/DUP, PPC ROM
11 REAL ESTATE, CCD
12 MACHINE
13 THERMAL
14 NAVIGATION, RAMBOX A
15 PETROLEUM, MC EPROM
16 PETROLEUM

17 PLOTTER, NFCROM
18 PLOTTER
19 CLINICAL, SECURITIES, AVIATION, STRUCTURAL ANAL.
20 PPC ROM
21 DATA LOGGER, ASSEMBLER 3
22 IL DEVELOPMENT, ADVANTAGE
23 EXTENDED I/O
24 IL DEVELOPMENT, ADVANTAGE
25 EXTENDED FUNCTIONS
26 TIME MODULE
27 WAND
28 HP-IL
29 PRINTER
30 CARD READER, RAMBOX 1B
31 DATA LOGGER

In order to store several programs in your RAMBOX, you need to proceed in the following order:

1) Initialize a page (9-15) using the function **INITPG** (see description of functions).
2) Load your program using the function **LDPGM**.

If you wish to change a program which is present in your RAMBOX, you need to copy this program into the main memory, using the **COPY** function of your HP-41 (see owner's manual for HP-41). Now you can edit your program and then load it back into the RAMBOX.

Important Advice

Since the operating system of the RAMBOX only allows for the erasing of the last program on a page, it is advisable to always reserve one page for copying. This is advisable because all programs of one page will have to be erased, even if you only want to erase the first program of a page. Therefore you should only put programs that will not be changed anymore at the beginning of a page. Furthermore it is advisable to end all programs in the RAMBOX with an **RTN** instead of an **END** and to tie them together as large programs, since this makes it easier to alter programs. For example, if you want to load a new program into the RAMBOX, you will first have to copy the last program of your RAMBOX into the calculator. Then you tie both programs together into a large program by substituting the **END**, which is located between the programs, with an **RTN**. Please note that it is important to avoid conflicts, which can occur when using the same numeric **LBL**s in both programs. Now the old program in the RAMBOX can be erased (using **CLLSTFL**) and

then the new program - increased by one more program - can be loaded. This makes it possible to fill one page of the RAMBOX with two large programs (which should actually consist of many small part programs). All part programs of the last large program can easily be edited.

1.2.3 EPROM Burning Service

If you want to offer your program package which you have tested with your RAMBOX to a greater circle of customers, we can burn your 4k blocks on EPROM plug-in modules the size of a standard plug-in module. This enables you to offer your programs in form of a module. A slightly cheaper solution is offered by the use of the EPROMBOX, which is put into the same case as your RAMBOX. It can be equipped with the usual EPROMs sold. The capacity of an EPROM module is 8-16k, that of the EPROMBOX is 4-32k.

If you have questions about EPROM burning services, use, USER-code- and M-Code programming, just contact us, there will always be someone around to help you:

W&W Software Products
Postfach 800133
Im Aehlemaar 20
D-5060 Bergisch Gladbach 2
West - Germany
Tel.: 02202 / 85068

W&W Software Products
1850 East 17th Street, Suite 102
Santa Ana, CA 92701
U.S.A.

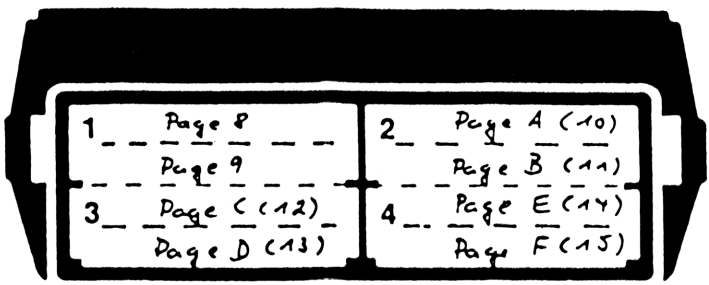
1.3 ROM MEMORY STRUCTURE

The HP-41 can manage a ROM-memory area of a maximum of 64kB. These 64kB are divided into sixteen 4kB pages, which are numbered from hex 0-F. These pages are used as follows:

Page	Used for
0	Operating system (System ROM 0)
1	Operating system (System ROM 1)
2	Operating system (System ROM 2)
3	Not used by HP-41C and CV. Extended operating system of the HP-41 CX. (System ROM 3)
4	Service module or disabled IL printer.
5	Used for the TIME module in the HP-41 C and CV. Extended operating system in the CX (system ROMs 5a and 5b with bank switching).
6	Reserved for the printer ROM.
7	Reserved for the HP-IL module (note: the printer ROM is contained in the HP-IL module, but can, using a certain switch, be put on the address area 4 and therefore be switched off)
8	PORT 1 lower 4 kByte
9	PORT 1 upper 4 kByte
A (10)	PORT 2 lower 4 kByte
B (11)	PORT 2 upper 4 kByte
C (12)	PORT 3 lower 4 kByte
D (13)	PORT 3 upper 4 kByte
E (14)	PORT 4 lower 4 kByte
F (15)	PORT 4 upper 4 kByte

The read/write memory of the main and extended RAM expansion modules is managed in a different manner from ROM, and is not addressed to the port which it occupies; thus it is possible to have all of your memory modules built into the HP-41, leaving the four I/O ports free for application pacs. One could even have the CCD-module installed internally and addressed to port 3, leaving that port free for the HP-IL or printer modules; since the addresses of these ROMs are fixed internally by the HP-41, other modules may be inserted in the "ports" they occupy if special electronic modifications are arranged to enable this (for more information on this subject contact any W&W Software Products division).

The following picture demonstrates how the address blocks in your HP-41 are arranged:



1.4 I/O BUFFER

I/O usually stands for input and output. Here it means "dialog with RAM memory while avoiding the operating system". The I/O buffer may be used by plug-in modules. Some modules for example the TIME and CCD-Module, each construct an I/O buffer and manage it independently. The I/O buffer appears to the operating system as a closed register block.

Each I/O buffer is identified by the base register which is the lowest numbered register in the block. The four nybbles at the very right contain the most important information:

Base register: hex *ii ll*.....

A copy of the buffer ID number is contained in nybbles 12 and 13 (*ii*), an ID number between hex 1 and hex E (hex F is reserved for the key assignment registers) is allowed. The two nybbles *ll* indicate the length of the buffer in registers.

When switching on the calculator, the operating system searches for buffers. If one is found, it erases the ID in nybble 13. Then it jumps to the register above the buffer and checks if there is another buffer there and so on. If no more buffers are found, it branches into the plug-in modules, which can reclaim their buffers by restoring nybble 13. Once all of the modules are checked we branch back into the operating system, which now erases the unclaimed buffers using a special PACK-I/O routine, and packs the buffer registers. This elucidates why, if the appropriate module is not plugged in when the calculator is turned on, the I/O buffer is erased.

2 GENERAL AID FUNCTIONS

2.1 BUFLNG? (Buffer Length)

The function **BUFLNG?** calculates the number of registers occupied by an I/O buffer. This function serves as a useful aid for working with buffer files, as is explained in 4.14.

Input parameter:

X register: *Buffer ID*

2.2 KEYAS? (No. of KEY assignment Registers)

This function calculates the number of registers occupied by key assignments. Chapter 4.11 (**CRFLKEY**) shows handy user's examples for **KEYAS?**.

2.3 PG? (Page Contents)

PG? now makes it possible to receive program-steered information by way of the contents of any 4K block. After entering the page # in the X register, the name of the stated page is put down in the ALPHA register; the X register contains a number in the format of XX.NN in which XX states the respective XROM # and NN states the number of functions contained in the page (the name of the page is counted as a function, when determining the number of functions).

Input parameter:

X register: *Page #*

Example for use:

Presuming that the operating system of your RAMBOX is still in Page 8 (state of delivery), it is possible to execute the following simple example:

Input	Output	
8 **PG?**	X register:	30.35
	ALPHA reg.:	-RAMBOX 1B

This means that "-RAMBOX 1B" is now on page 8. It has the XROM # 30 and contains 35 functions.

2.4 FNC? (Description of Function)

With the aid of **FNC?** it is now possible to get information about separate functions of a page - program steered. Using a steering digit of the sort XX.NN (in which XX=XROM # and NN=Function number) in the X register we get the decimal byte values in the X register, after executing **FNC?**. In the Y register we get the hexadecimal byte values as an ASCII-string, in the Z register we get the XROM # and the function number incremented by 1; and finally the ALPHA register

shows us the XROM # on the positions 1-5 followed by 2 spaces, the function name on the positions 8-18 once more followed by two spaces; and last, but not least, the address on which the function of the respective page begins is shown on positions 21-24.

If **FNC?** is used in a program, the execution of the next step is dependent on whether the function specified in the X-register is existent or not; if this function does not exist, the next program step is skipped.

Input parameter:

X register:	*XX.NN*	*XX* = XROM # *NN* = Function #

Output:

X register:	*bbb.bbb*	Byte values decimal
Y register:	*"BB.BB"*	Byte values hexadecimal
Z register:	*XX.(NN+1)*	*XX* = XROM # *NN* = Function #
ALPHA reg.:	*"XX.NN Name... AAAA"*	*AAAA* = Address

Example for use:

The following program "PRFAT" prints a list of all functions contained in a certain page - on an HP-ThinkJet printer. (This program is part of the RAMBOX operating system.)

Program "PRFAT"

```
01▸LBL "PRFAT"◂
02 "◂&k0S◂"◂
03 SF 25◂
04 DATE◂
05 FIX 6◂
06 SF 25◂
07 ADATE◂
08 CF 25◂
09 ACA◂
10 "◂&k1S      "◂
11 ACA◂
12 "PAGE ?"◂
13 PROMPT◂
14 PG?◂
15 "■◂&k0S◂◂"◂
16 PRA◂
17 "XROM#  Function"◂
18 ACA◂
19 "     Adr.  Byte"◂
20 ACA◂
21 "(H)  Byte(D)◂◂="◂
22 PRA◂
23 SF 28◂
24 FIX 3◂
25 INT◂
26▸LBL 05◂
27 FNC?◂
28 GTO 00◂
29 GTO 01◂
30▸LBL 00◂
31 ACA◂
32 CLA◂
33 "    "◂
34 ARCL Y◂
35 "■    "◂
36 ARCL X◂
37 PRA◂
38 RDN◂
39 RDN◂
40 GTO 05◂
41▸LBL 01◂
42 "◂"◂
43 ACA◂
44 BEEP◂
45 END◂
```

Printout with the program "PRFAT"

09/15/1986 -RAMBOX 1B

XROM#	Function	Adr.	Byte(H)	Byte(D)
30.00	-RAMBOX 1B	8058	A7.80	167.128
30.01	BUFLNG?	894E	A7.81	167.129
30.02	CLLSTFL	848E	A7.82	167.130
30.03	CLPG	8D84	A7.83	167.131
30.04	CLRFL	8ADC	A7.84	167.132
30.05	COPYPG	8111	A7.85	167.133
30.06	CRDIR	8192	A7.86	167.134
30.07	CRFLBUF	897F	A7.87	167.135
30.08	CRFLDTA	8967	A7.88	167.136
30.09	CRFLKEY	8973	A7.89	167.137
30.10	ENDPG	872E	A7.8A	167.138
30.11	FNC?	888C	A7.8B	167.139
30.12	FRBYT?	8225	A7.8C	167.140
30.13	GTBUF	8AF3	A7.8D	167.141
30.14	GTKEY	8B99	A7.8E	167.142
30.15	GTREG	8B44	A7.8F	167.143
30.16	GTREGX	8B5A	A7.90	167.144
30.17	GTREGXY	8B63	A7.91	167.145
30.18	INITPG	805F	A7.92	167.146
30.19	KEYAS?	895B	A7.93	167.147
30.20	LDBUF	8A6D	A7.94	167.148
30.21	LDKEY	8A51	A7.95	167.149
30.22	LDPGM	8270	A7.96	167.150
30.23	LDREG	8A96	A7.97	167.151
30.24	LDREGX	8AAA	A7.98	167.152
30.25	LDREGXY	8AB3	A7.99	167.153
30.26	PG?	880B	A7.9A	167.154
30.27	PGSUM	86C1	A7.9B	167.155
30.28	PTCT	862B	A7.9C	167.156
30.29	READPG	8EC2	A7.9D	167.157
30.30	SETPRV	8624	A7.9E	167.158
30.31	UNPTCT	8633	A7.9F	167.159
30.32	WRTPG	812D	A7.A0	167.160
30.33	XQ>XR	8758	A7.A1	167.161
30.34	'PRFAT	8DBD	A7.A2	167.162

2.5 XQ>XR (Transform XEQ to XROM)

The function **XQ>XR** changes all XEQ orders in a program for which a corresponding global LBL (ALPHA LBL) is found in a ROM or in the RAMBOX into XROM orders.

In this action the stating program as well as the program being worked on can be in the main memory as well as any page of the RAMBOX (for this also see the comments for **LDPGM** in chapter 4.1). If the program being worked on is in the RAMBOX, the change does not cause any space to be saved (saved bytes are substituted by zero-bytes), but a considerable advantage of speed - compared to the version that has not been worked on - is nevertheless obtained.

Input parameter:

ALPHA reg.: *Name of the program to be worked on*

2.6 CRDIR (Create Directory Entry)

This function allows for the access of the entire memory area of a discette for HP-9114 disc drive.

Normally the usable area is limited to 130 kBytes by the IL-operating system of the HP-41, but with the aid of **CRDIR** the full capacity of a discette can be used. If one intends to put a file of any sort on a discette, which would ordinarily exceed the limit of 130 kByte, one has to state the needed file size in registers in the X register. After execution of **CRDIR** there is a catalog on the discette which can now be used to copy a file.

Input parameter:

X register: *Number of registers needed*

2.7 SETPRV (Set Private)

The function **SETPRV** makes it possible to give a program the private status - without the detour of an external memory medium.

For this there are three different modes:

1. Program is in the main memory
   The program has the private-protection, but can, after being copied into the RAMBOX using **LDPGM**, be loaded back into the main memory using **COPY**, and of course will still have the private-status.
2. Program is in the RAMBOX
   The program gets the private-protection and can not be loaded back, even with **COPY**.
3. The program was protected with the private status first in the main memory and then once more in the RAMBOX. The same result as in 2 is obtained.

Input parameter:

ALPHA reg.: *Name of the program to be protected*
(If the ALPHA register is empty, the current program is protected.)

3 MANAGEMENT FUNCTIONS

3.1 CLPG (Clear Page)

CLPG is an aid function which allows for the complete erasing of a 4kByte block, i.e., the page specified in the X register is completely filled with 0-bytes.

Hint:
This function does not do any safety-questioning; i.e., if the function is accidentally executed with a valid page # in the X register, the entire content of this 4kByte block is lost.

Input parameter:

X register: *Page #*

3.2 INITPG (Initialize Page)

The function **INITPG** opens a page with which one can work from now on, until the first execution of **ENDPG** (see chapter 3.8). It requires the page number in the ALPHA register, the page # in the X register, and the selected XROM number in the Y register. **INITPG** erases - just like **CLPG** - the entire specified 4kByte block first, and then enters the data specified in the X-, Y-, and ALPHA registers.

Hint:
The function does not do any safety-questioning, i.e., if the function is accidentally executed with a valid page # in the X register, the entire content of this 4kByte block is lost.

Input parameters:

X register:	*Page #*
Y register:	*XROM #*
ALPHA reg.:	*Name of the page* (In this only the 11 left hand signs up to the first comma are taken into consideration. If the ALPHA register is empty, "-" is entered as a name.

3.3 FRBYT? (Free Bytes in Page)

FRBYT? gives you the number of bytes still left to use in a page specified by the contents of the X register.

Input parameter:

X register:	*Page #*

User's example:

Now execute the following program in order to get to know the uses of the functions **INITPG** and **FRBYT?** (note that page 10 is erased during this). First remove all ROM-modules from your calculator and then carry out the following instructions:

ALPHA "-RAM 1A" ALPHA
9 ENTER 10
XEQ ALPHA "INITPG" ALPHA

Now execute **CAT 2** and after the display of the functions of the RAMBOX you will see the catalog statement of the page that was just initialized:

"-RAM 1A"

Now execute:

10
XEQ ALPHA "FRBYT?" ALPHA

X register will show results:

4056.00

This gives you 4056 bytes in page 10 (or 670 data registers) for the storing of data and programs.

3.4 COPYPG (Copy Page)

With the help of **COPYPG** one can copy the complete contents of a page onto another page of the RAMBOX.

Input parameters:

X register: *Goal page #*
Y register: *Initial page #*

3.5 WRTPG (Write Page)

The function **WRTPG** allows for the copying of any page onto a mass memory unit. For this the name which the file is supposed to have on the tape (or any other mass memory unit) is written into the ALPHA register, and the page # is written into the X register. The file occupies 640 registers on the mass memory unit and has the type 7 (this type is shown as "?" when executing **DIR**).

Hint:
The function **WRTPG** is compatible with the function SAVEROM of the ERAMCO-MLDL.

Input parameters:

X register: *Page #*
ALPHA reg.: *File name*

3.6 READPG (Read Page)

Using **READPG** it is possible to read a 4 kByte block - copied onto a mass memory unit using **WRTPG** - back into the RAMBOX. For this the name of the file on the cassette in the ALPHA register and the number of the page to be loaded into the display are stated in the X register.

Hint:
The function **READPG** is compatible to the function **GETROM** of the ERAMCO-MLDL.

Input parameters:

X register: *Page #*
ALPHA reg.: *File name*

3.7 PGSUM (Pagesum)

The function **PGSUM** calculates the checksum of a page and writes this into the address XFFF of the respective page. With the respective page # in the X register the following display is shown during the execution:

"PG: NN RR-RR"

in this NN stands for the page # and RR-RR stands for the ROM-revision.
Once the calculation has ended, the following display is shown:

"RR-RR INTACT" or "RR-RR BROKEN"

With this **PGSUM** does three different tasks:

1. **PGSUM** allows for the display of the ROM-revision of your own and also foreign modules.
2. **PGSUM** allows for the calculation and the input of the checksum of the respective page.
3. **PGSUM** compares the newly calculated checksum with the checksum already stated and shows "INTACT" when the results are the same, it shows "BROKEN" when the results are different, i.e., the possibility to check if data in a module or in one page of the RAMBOX has changed, is given. Due to this the first execution of **PGSUM** - after data has been changed in the respective page - will yield the result "BROKEN". Nevertheless, the new checksum is entered even in this case, which can easily be checked by renewed execution of the function.

Input parameter:

X register: *Page #*

3.8 ENDPG (End Page)

ENDPG ends the work on a page. During this the contents of the ALPHA register are entered as a ROM-revision, and the checksum is calculated, which draws forth the same display as in the execution of **PGSUM** (see 3.7).

Hint:
The function **ENDPG** should only be carried out once the work on one page is completely finished. A renewed access to a page closed with **ENDPG** is not possible anymore. Every attempt to get access to this page with a function storing data will yield the error message "PAGE CLOSED".

Input parameter:

X register:	*Page #*
ALPHA reg.:	*ROM-Revision* (during this only the 4 left hand signs up until the first comma are regarded. If the ALPHA register is empty, "----" is entered as the ROM-revision.

4 FILE FUNCTIONS

4.1 LDPGM (Load Program)

LDPGM enables you to load any user's programs which were specified by their name in the ALPHA register into the page mentioned in the X register. If the ALPHA register is empty, the current program is copied onto the chosen page. This gives you the possibilty to copy several of the programs to be copied into the main memory, to erase the ALPHA register, to position the calculator onto the respective program - using **CAT 1** - and to load it into the RAMBOX without having to explicitly mention the name for every new loading process.

Input parameters:

X register:	*Page #*
ALPHA reg.:	*Name of the program to be loaded* (if the ALPHA register is empty, the program on which the calculator is positioned is loaded.)

4.2 CLLSTFL (Clear last File)

The function **CLLSTFL** erases the last program contained in the page or the last data-, BUFFER- or KEY-file (the meaning of these file types will be explained in the following

chapters). If the file to be erased has been protected with **PTCT**, it has to be safety-released before erasing, using **UNPTCT**.

Input parameter:

X register: *Page #*

4.3 CRFLDTA (Create Data-File)

CRFLDTA enters a data file with the size specified in the Y register into the page specified in the X register. Such a file can now - comparable to a data file in the X-Function- resp. X-Memory module - be written and read with the functions explained below:

Hint:
Calling a data file with **XEQ'** "file name" will be without any effect.

Input parameters:

X register: *Page #*
Y register: *File size* (max. 670 registers)
ALPHA reg.: *File name*

4.4 LDREG (Load Registers)

The function **LDREG** copies all existent data registers into the data file specified in the ALPHA register. Should you try to load more registers than the file can take up, the error message "END OF FILE" will be displayed.

Input parameter:

ALPHA reg.: *File name*

4.5 LDREGX (Load Registers by X)

LDREGX works similar to **LDREG** with the exception that one has to put a control number of the type bb.eee into the X register, which determines which register block is to be copied into the data file.

Input parameters:

X register: *bbb.eee* *bbb* = first register to be copied
eee = last register to be copied

ALPHA reg.: *File name*

4.6 LDREGXY (Load Registers by X and Y)

The function **LDREGXY** allows for an even more extended control of the data copying process as **LDREGX** (see 4.5). Contrary to **LDREGX**, this function expects an extra parameter in the Y register, which determines from which data register in the data file the data from the main memory is supposed to be put down.

Input parameters:

X register:	*bbb.eee*	*bbb* = first register to be copied *eee* = last register to be copied
Y register:	*nnn*	*nnn* = Number of the first data register in the data file
ALPHA reg.:	*File name*	

4.7 GTREG (Get Registers)

GTREG is a function exactly opposite to **LDREG** (see 4.4) and copies the contents of a data file back into the data registers of the main memory.

Input parameter:

ALPHA reg.:	*File name*

4.8 GTREGX (Get Registers by X)

The function **GTREGX** is a function exactly opposite to **LDREGX** (see 4.5), and copies the contents of a data file into the registers of the main memory specified in the X register.

Input parameters:

X register:	*bbb.eee*	*bbb* = 1. register into which data is to be copied *eee* = last register into which data is to be copied
ALPHA reg.:	*File name*	

4.9 GTREGXY (Get Registers by X and Y)

Analagous to **LDREGXY** (see 3.6), data from a data file - starting from the file register specified in the Y register - can be copied back - using **GTREGXY** - into the data registers of the main memory, which are specified in the X register.

Input parameters:

X register:	*bbb.eee*	*bbb* = 1. register into which data is to be copied *eee* = last register into which data is to be copied.
Y register:	*nnn*	*nnn* = 1. register in the data file from which on the copying into the main memory is to start.
ALPHA reg.:	*File name*	

4.10 CLRFL (Clear File)

CLRFL makes it possible to completely erase the contents of a data file.

Input parameter:

ALPHA reg.: *File name*

4.11 CRFLKEY (Create KEY-File)

The function **CRFLKEY** allows for the making of a KEY file into which the key assignments can be put. The file has a maximum size of 42 registers. Before making a new KEY file, one can use the function **KEYAS?** to find out which file size is necessary to store the currently existing key assignments.

Hint:
Calling a KEY-file with **XEQ** "File name" will have no effect.

Input parameters:

X register: *Page # in which the file is to be made.*
Y register: *File size*
ALPHA reg.: *File name*

4.12 LDKEY (Load KEY-Assignments)

BLDKEY transfers the currently valid key assignments into the specified KEY-file.

Hint:
Existing key assignments belonging to user's programs are not touched by **LDKEY** and **GTKEY.**

Input parameter:

ALPHA reg.: *File name*

Example of use:

The following example establishes a file with the name of "KEY1" in page 10 and transfers the existing key assignments into this file.

XEQ ALPHA "KEYAS?" ALPHA	required number of registers to X.
10	The KEY file is to be put down in page 10.
ALPHA "KEY1" ALPHA	File name "KEY1"
XEQ ALPHA "CRFLKEY" ALPHA	Establish file
XEQ ALPHA "LDKEY" ALPHA	Transfer of the key assignments to the KEY file.

4.13 GTKEY (Get KEY Assignments)

GTKEY erases the existing key assignments and copies the key assignments stored in a KEY file into the calculator.

Hint:
Existing key assignments belonging to user programs are not touched by **LDKEY** and **GTKEY**.

Input parameter:

ALPHA reg.: *File name*

4.14 CRFLBUF (Create Buffer File)

The function **CRFLBUF** establishes a BUFFER file in the page specified in the X register, into which the contents of I/O buffers can be put. The maximum size of a buffer file is 255 registers.

Hint:
Trying to call a buffer file using **XEQ'** "file name" will be without any effect.

Input parameters:

X register: *Page #*
Y register: *File size*
ALPHA reg.: *File name*

4.15 LDBUF (Load Buffer)

LDBUF transfers the contents of an I/O buffer into a buffer file.

Input parameters:

X register: *Buffer ID*
ALPHA reg.: *File name*

4.16 GTBUF (Get Buffer)

GTBUF copies the contents of a buffer file into an I/O buffer. Should there already be an I/O buffer with the same ID, its old contents will be erased and substituted by the new ones; on the other hand should there not exist a buffer with the same ID, one is established during the execution of **GTBUF.**

Input parameter:

ALPHA reg.: *File name*

4.17 PTCT (Protect a File)

PTCT can be used to protect a file of any sort against unwanted overwriting and erasing.

Input parameter:

ALPHA reg.: *File name*

4.18 UNPTCT (Unprotect a File)

UNPTCT undoes the file protection set with **PTCT**. After execution of **UNPTCT** all files can once again be written on, erased and cross faded as before.

Input parameter:

ALPHA reg.: *File name*

5 ERROR MESSAGES

Error Message	Function	Explanation
ALPHA DATA	all functions expecting numerical parameters	Alpha characters are in a register where a number is expected
DATA ERROR	BUFLNG?	Buffer ID<1 or Buffer ID>14
	CRDIR	Number of given registers <1 oder >9999
	CRFLBUF	File size <1 or >255
	CRFLDTA	File size <1 or >670
	CRFLKEY	File size <1 or >42
	FNC?, INITPG	XROM # <=0 or >31
	all functions that work on a given page	Page # <5

DUB NAME	CRFLBUF, CRFLDTA, CRFLKEY	a file with the given name already exists
DUB XROM#	INITPG	the XROM # used is already present in another page
END OF FILE	LDBUF, LDKEY, LDREG, LDREGX, LDREGXY	more registers than a file can take in were tried to enter
FAT FULL	CRFLBUF, CRFLDTA, CRFLKEY, LDPGM	There is not enough space in the function address table of the specified page to be able to store the required table entries

FL NOT FOUND	GTBUF, GTKEY, GTREG, GTREGX, GTREGXY, LDBUF, LDKEY, LDREG, LDREGX, LDREGXY	Specified file does not exist
FL PROTECTED	CLLSTFL, LDBUF, LDKEY, LDREG, LDREGX, LDREGXY	The specified file is protected and can not be manipulated this way
ILLEGAL CHR	CRFLBUF, CRFLDTA, CRFLKEY, ENDPG, INITPG	Using ALPHA characters with the ASCII codes <=0F, 2E, 3A or >=66 in the specified name is not permitted
INIT ERR	LDPGM	The chosen page has not yet been initialized

all IL error messages	CRDIR, READPG, WRTPG	see HP-IL handbook
NO ACCESS	all functions causing storing or erasing operations in a page	working on the operating system is not permitted
NO ALPHA LBL	LDPGM	the given program does not contain a global LBL under which it could later be found
NO BUFFER	LDBUF	no buffer with the given ID present
NO FILE	CLLSTFL	no more files in the specified page
NO HPIL	CRDIR, READPG, WRTPG	No interface loop present

NO KEYS	LDKEYS	no key assignments present
NO LBL NN	LDPGM	the given program does not contain a numerical LBL NN, even though a GTO or XEQ order is supposed to branch into it
NO NAME	all functions working on files	the file name was not specified
NO RAM	all functions causing storing or erasing operations in a page	not possible to write on the chosen page
NONEXISTENT	FNC?	there exists no function with the given XROM #
	GTREG, GTREGX, GTREGXY, LDREG, LDREGX, LDREGXY	the given data registers are not present

	PG?	the page called is empty
	PTCT, UNPTCT SETPRV, XQ>XR	there exists no file with the given name there exists no program with the given name
	all functions that work on a given page	Page # >15
PAGE CLOSED	FRBYT?, all functions causing sto- ring or erasing operations in a page (exept INITPG and CLPG)	the specified page has already been closed with ENDPG and can not be worked on anymore with the executed funktion
PACKING-TRY AGAIN	LDPGM	the chosen program did not have an END of its own. Trying to annex one did not work, since there is no more space left in the memory

RAM	PTCT, UNPTCT	the mentioned program is in the main memory
ROM	LDPGM, SETPRV	the mentioned program is in the ROM
ROM FULL	CRFLBUF, CRFLDTA, CRFLKEY, LDPGM	there is not enough space left in the page to enter the file, i.e. to store the program

6 WARRANTY

The W&W Software Products RAMBOX is guaranteed for ninety (90) days from the date of purchase. Should a RAMBOX prove to be defective within ninety (90) days, return it to us (at your expense) and we will replace any defective unit with a new one. Should any part of your RAMBOX malfunction after the above mentioned time, you may call any one of our W&W Software Products divisions, prefereably the one you acquired the RAMBOX from. They will give you an estimate of costs for repair or replacement, a return authorization number and shipping instructions.

THE FOREGOING LIMITED WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE.

The material contained in the RAMBOX instructions is supplied for the convenience of RAMBOX owners and is supplied without representation or warranty of any kind. In any event, the liability of W&W Software Products or its marketing agents for damages, regardless of the form of action, shall not exceed the charges paid by you for the RAMBOX, and by acceptance of the RAMBOX you specifically agree that neither W&W Software Products or its marketing agents shall be liable to you for any loss incurred while using the RAMBOX, or for any clain or demand against you by any other person arriving out of use of the RAMBOX. In no event will W&W Software Products or its marketing agents be liable to you or any other party for consequential damages even if you have advised us of the possibility of such damages.

W&W Software Products GmbH
Im Aehlemaar 20, Postfach 800 133
5060 Bergisch Gladbach 2, Tel.: 02202/85068