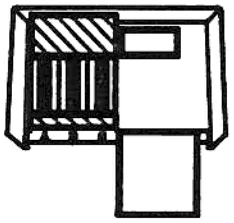


W&W SOFTWARE PRODUCTS

OWNER'S MANUAL

32 kiloBYTE RAM BOX

for the HP-41 Family of Handheld Computers

English Edition by

Stephan B. Abramson, Ph. D.

-- November 1986 --

W&W SOFTWARE PRODUCTS

1735B NE Seavy Avenue, Corvallis, Oregon 97330 USA

CodeSmith (503) 752-0634 Thursday 1st January 1987

Dear Customer,

I am delighted to finally send you your 32K RAM Box Owners manual. Please note that this manual is the sole work of Stephan Abramson who not only translated it from the german original but also produced all the typesetting and artwork, in painstaking detail, on a high resolution dot matrix printer; the cover on an HP plotter. This was done in his free time which, as is the case with any busy professional, especially during these times of great change, is sparse indeed. I trust you will forgive us the delay in getting this manual to you.

W&W Information:

W&W Software Products is a West German Company run by Wilfried Koetz. The company manufactures the CCD ROM, 32k RAM Box, 32K EPROM Box and other HP-handheld related electronic products. They also distribute the PAC Screen and Grabau HP-IL graphics video interfaces and other products throughout Europe, and internationally.

I, Jeremy Smith, set up the US branch of the company in December 1985. Effectivly I was the manufacturers representative for W&W in the USA. In June 1986 I became employed by CMT and, to avoid conflict of interest, among other things, released my interest in W&W to S.O.S Company run by David White.

S.O.S Company do have some W&W products, which are available directly from them or their main distributor EduCalc (All addresses given below). However, S.O.S Company intends to discontinue representing W&W and, therefore, all future enquiries should be directed to W&W in West Germany. Since they are continuing to develop exciting new products, and have quite an extensive product range it is recommended that you write to them for one of their product catalogs. Still the most comprehensive source of HP-handheld products in this country, including W&W products, is EduCalc whose catalog is well worth the phone call.

Best Wishes and a Happy New Year

Jeremy Smith

Jeremy Smith

Wilfried Koetz	David White	Jim Carter	Old address:
W&W Software Products GmbH	S.O.S Company	EduCalc	Jeremy Smith
Im Aehlemaar 20	1850 E 17th Street	27953 Cabot Road	2056 maple Ave
Postfach 800 133	Suite #102	Laguna Niguel	Costa Mesa
D-5060 Bergisch Gladbach 2	Santa Ana	California 92677	CA 92627 USA
West Germany	California 92701	(714) 582-2637	(714) 626-1935
Phone: 02202/85068	(714) 558-1806	(800) 633-2252	

Table of Contents

1. INTRODUCTION

The first section of this chapter explains the procedure for installing the RAM BOX and the options available for addressing sections of its memory; Section 2 presents the fundamentals of its operation. The remaining sections give an overview of the protocol used by the HP-41's operating system to address read-only memory (ROM) and to organize I/O (input/output) buffers.

1.1 Installation

***An Important Note of Caution**—Be sure that your HP-41 is turned off before you insert or remove the RAM BOX; otherwise you may cause a static discharge which could damage the computer or the peripheral.*

Before first installing the RAM BOX, remove the Extended Functions module and ***all*** application ROMs, from your HP-41. You need not remove a TIME module or such RAM extensions as the QUADRAM or Extended Memory modules.

Your RAM BOX has a storage capacity of 32 kBytes (32768 Bytes). Of this total 4 kBytes (4096 Bytes) are occupied by the operating system provided by W&W Software. This system includes 32 new functions (written in microcode) designed to facilitate your use and control of the RAM BOX.

After you insert your RAM BOX for the first time, turn the computer on, enter **PROGRAM** mode and key in the following routine:

```
01*LBL "LP"      06 FS?C 25         11 PROMPT
02 8.015         07 GTO 02          12*LBL 02
03*LBL 01        08 ISG X           13 "COMPLETED"
04 SF 25         09 GTO 01          14 PROMPT
05 LDPGM         10 "NOT LOADED"    15 END
```

*Note that **LDPGM** (line 05) is a function included in the operating system of the RAM BOX.*

Now, to load any program from the main read/write memory (RAM) of your HP-41 into the RAM BOX, simply key one of its global (*i. e.*, **ALPHA**) labels into **ALPHA** and execute "LP". If the loading of the program is proceeding normally, the HP-41 will successively display these messages:

```
PACKING
LOADING
COMPILING
COMPLETED
```

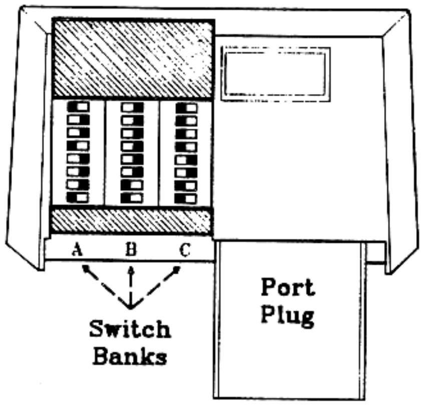
By executing **CAT 2**, you can verify that the program you have loaded is contained within the read/write memory (RAM) of the RAM BOX. Once you have stored all the programs you wish in the RAM BOX, you may clear the programs from the main memory of your HP-41 without affecting the status of the programs loaded in the RAM BOX. Moreover, these programs may still be accessed as easily as if they were still in the main memory RAM of the computer: they may be executed using the familiar keystroke sequence

XEQ ALPHA "(global label)" **ALPHA**

Anytime you wish to restore one of these programs to the HP-41's main memory RAM, simply use the HP-41's resident COPY function.

Your RAM BOX contains a lithium battery with a minimum life expectancy of 5 years. This battery maintains the integrity of programs and data loaded in your RAM BOX, whether or not the peripheral is plugged into the HP-41. Of course, while the RAM BOX is inserted in your HP-41, its circuits are common with those of the computer. Data in the RAM BOX are thus maintained indefinitely in this configuration, as long as the batteries in the HP-41 remain live. ***However, we recommend keeping backup files on disks, cassettes, or magnetic cards, of all programs and data you routinely store in the RAM BOX, to guard against the inadvertent loss of important information.***

The memory of the RAM BOX is partitioned into eight "blocks" called "pages", each of which contains 4 kBytes (4096 bytes). Operation of and access to these pages is controlled by the settings of 3 banks of dual in-line polarity (DIP) switches located on the underside of the RAM BOX (see below).



RAM BOX (Bottom View)

These banks, each of which contains eight switches, are designated A, B and C, as shown in the Figure above. They are oriented so that the ON position is at the left side of each switch. Bank A sets the binary codes used to position two 4 kByte pages optionally to any 4 kByte region within the 64 kByte addressable range. Switches A1 - A4 define the address of the first page and switches A5 - A8 set the address for the second page according to the coding scheme illustrated in the Table on the following page.

To illustrate the use of these switches, we shall address one optional RAM page to block 7 and the second to block 15 (E_h) To accomplish this we set the switches in bank A as follows:

Switch	A1	A2	A3	A4	A5	A6	A7	A8
Position	off	on	on	on	on	on	on	off
Block Address	7				E_h (15_d)			

Address*	Switch Number				Address*	Switch Number			
	A1	A2	A3	A4		A5	A6	A7	A8
0	off	off	off	off	0	off	off	off	off
1	off	off	off	on	1	off	off	off	on
2	off	off	on	off	2	off	off	on	off
3	off	off	on	on	3	off	off	on	on
4	off	on	off	off	4	off	on	off	off
5	off	on	off	on	5	off	on	off	on
6	off	on	on	off	6	off	on	on	off
7	off	on	on	on	7	off	on	on	on
8	on	off	off	off	8	on	off	off	off
9	on	off	off	on	9	on	off	off	on
10 (A)	on	off	on	off	10 (A)	on	off	on	off
11 (B)	on	off	on	on	11 (B)	on	off	on	on
12 (C)	on	on	off	off	12 (C)	on	on	off	off
13 (D)	on	on	off	on	13 (D)	on	on	off	on
14 (E)	on	on	on	off	14 (D)	on	on	on	off
15 (F)	on	on	on	on	15 (E)	on	on	on	on

*For address numbers > 9, the hexadecimal notation is given in parentheses to the right of the decimal representation.

The switches in bank B determine the write-protection status of individual pages. When the switch for a given page is ON (write-*enabled*), the page is treated as RAM, and programs and data may be stored or cleared on it. If the switch for a page is OFF (write-*disabled*), however, that page looks like ROM to the HP-41: its contents cannot be inadvertently cleared or overwritten by other information.

Bank C is used to place individual pages "on-" or "off-line" (switch them into and out of the HP-41's active circuitry). ***Note that switches B1 and C1, and B2 and C2 correspond to the RAM "pages" whose addresses are determined using switches A1 - A4 and A5 - A8, respectively.***

To illustrate the use of these switch banks as a complete memory control system, let us position the two optional pages in blocks 8 and 9 with write-protection, switch pages A and B_h off and use pages C_h, D_h, E_h and F_h without write-protection. To establish this operational status we set:

Bank\Switch	1	2	3	4	5	6	7	8
A	on	off	off	off	on	off	off	on
B	on	on	off	off	off	off	off	off
C	on	on	off	off	on	on	on	on

1.2 General Remarks

If the first two pages of your RAM BOX are set to addresses 8 and 9 (*i. e.*, the settings with which they are usually shipped from the factory), you will have seven 4 kByte blocks available for loading programs. These seven blocks will have the addresses 9_d - 15_d (9_h - F_h), since block 8 is occupied by the factory-installed operating system. In this configuration, your RAM BOX may be used to simulate seven 4 kByte application modules, such as the STAT or MATH PACs. In contrast to commercially available ROMs, however, the contents of your RAM BOX may be changed by you at any time.

1.2.1 Page Headers

We recommend using descriptive headers to initialize the 4 kByte pages in your RAM BOX. Headers which categorize the contents of individual blocks by subject will facilitate the location of a given program using **CAT 2**. The headers you use to categorize your ROM images may consist of any convenient alphanumeric string of $\leq$ 11 characters. It is usual to standardize page headers to the extent of always using a hyphen (ASCII code 45) as the *first* character; this helps to distinguish the headers from the included functions during a **CAT 2** listing. Some sample headers are:

-MICHAELROM	This is a general header
-MATRX W&W1 \ -VARNCE WW1 /	These are examples of headers for groups of related functions; note the use of contractions, spaces and revision numbers.

1.2.2 XROM Numbers

The HP-41 uses a two-digit number from 1 to 31 which we shall call the ROM ID number (*i. e.*, a binary number from 00001 to 11111) to distinguish among the several application ROMs; thus the same system of numbers is used to identify individual pages of the RAM BOX. Within each ROM or RAM page, a second number (the function ID) from 0 to 63, (*i. e.*, from 00 0000$_b$ to 11 1111$_b$) identifies the individual programs and/or functions (for simplicity, in the discussion which follows, we shall use the term *function* to refer to *both functions and programs* contained within ROMs) it includes. Thus the HP-41 can distinguish up to 64 different functions and programs (the first of which, numbered 00, is usually the ROM header) in a ROM or RAM page.

This scheme enables the HP-41 to generate an identification number for any ROM (or RAM page) function which combines the ROM and function identification codes and which thus completely defines the function in an operational sense. If, while you print a program listing, view a routine in **PROGRAM** mode or run a program, the HP-41 encounters a ROM function, the operating system searches CATalog 2 for a function with the specified identification number. If the system detects the requested ROM, the function is executed or its name is displayed or output. If the ROM is absent, however, the HP-41 displays or outputs in lieu of the function's name, a number derived from its identification code (note that if the function was to be executed, the calculator displays the error message "NONEXISTENT" and control over the computer's operation is returned to the keyboard). These numbers are displayed in the format "XROM aa,bb"; "XROM" is an acronym for e$\underline{x}$ternal $\underline{R}$ead-$\underline{O}$nly $\underline{M}$emory and aa and bb are two-digit decmial numbers. Because of this display format, the identification codes for ROM functions are called ***XROM numbers***.

Since the HP-41 uses the ROM ID to determine which ROM to search for the required function, the computer will not function properly with multiple ROMs in the system having the same ROM ID. You should therefore select ROM ID numbers for the address blocks in your RAM BOX wich differ from those of the application ROMs used in your HP-41 system. To assist you in selecting appropriate ROM IDs for your RAM BOX, we present a Table of ROM ID numbers used in application ROMs and peripherals which are currently available commercially.

XROM Number	Application ROM or EPROM		
01	MATH I		
02	STATISTICS I	DAVID ASSEMBLER	
03	SURVEYING		
04	FINANCIAL		
05	STANDARD PAC	ZENROM	PANAME (1)
06	CIRCUITS	ADVANCED FCNS.	
07	STRUCT'L. ANAL.(1)		
08	STRESS		
09	HOME MNGEMENT.	CCD ROM (1)	PANAME (1)
	CO-OP MODULE (1)		
10	AUTO START/DUPL'N.	GAMES I	PPC ROM (1)
	CO-OP MODULE (2)		
11	REAL ESTATE	CCD ROM (2)	
12	MACHINE	PPC ROM 2C	
13	THERMAL		
14	NAVIGATION	RAM BOX	
15	PETROLEUM (1)	MC EPROM	
16	PETROLEUM (2)		
17	PLOTTER (1)	NFCROM 1C	
18	PLOTTER (2)		
19	CLINICAL LAB.	SECURITIES	AVAIATION
	STRUCTL. ANAL. (2)	PPC EPROM 5A	
20	PPC ROM (2)		
21	CUSTOM 8K	ASSEMBLER 3	DATA LOGGER (1)
22	HPIL DEVEL. (1)	ADVANTAGE (1)	
23	EXTENDED I/O (1)		
24	HPIL DEVEL. (2)	ADVANTAGE (2)	
25	EXTENDED FUNCTIONS		
26	TIME		
27	OPTICAL WAND	EXTENDED IL	
28	HP-IL CONTROL AND MASS STORAGE FUNCTIONS		
29	PRINTER FUNCTIONS		
30	CARD READER		
31	CUSTOM 4k & 8k	DATA LOGGER (2)	DIAMOND TELLER

Before loading programs into your RAM BOX, you will need first to perform these operations:

1. Initialize one of the RAM pages (numbered 9 - 15_d) using the function **INITPG** (see the description in Chapter 3).
2. Use the function **LDPGM** to load your programs.

If you wish to alter a program which you have loaded into your RAM BOX you must first copy the program back into the main memory RAM of your HP-41 using the resident HP-41 function **COPY** (see your HP-41 Owner's Manual for further details if you are unfamiliar with the operation of this function). Once you have restored the program to main memory, you can edit it as you please, and then reload the revised version into the RAM BOX.

An Important Note of Caution—Since the operating system of the RAM BOX permits you to clear only the *last* program on a page, we recommend that you always reserve one page for the purpose of recopying programs. Note that in order to clear the first program on a given page, you must first clear all of the other programs on that page; it is thus prudent to load first on any given page the programs you are least likely to wish to alter.

We recommend further that programs you load into the RAM BOX be terminated with a **RTN** rather than an **END**. When you do this, individual routines on a RAM page are combined into a few very large programs which become easier to recall into main memory RAM and thus easier to revise than a number of short individual programs. Therefore when you wish to load a new program into the RAM BOX, first use **COPY** to retrieve the last program of the RAM page into main memory. Next, replace the **END** with a **RTN** to incorporate into the larger program the routine to be added. (*Caution:* **be sure that the programs you are merging contain no conflicting local labels**). Finally, the old last program may be cleared from the RAM BOX, and replaced by loading the new, expanded program. By exploiting all of main memory RAM for this process, you can fill an entire 4 kByte RAM page with two large programs assembled from a number of smaller routines, so that any of the partial programs may easily edited as a subroutine of the large last program.

1.2.3 EPROM Programming Service

If you have developed a series of programs that you have tested in your RAM BOX and would like to make available to a larger clientele, we can manufacture your custom 4 kByte blocks of program material in the form of plug-in EPROM modules using the same cases as commercially available application ROMs such as HP's MATH PAC. You will then be able to offer your software for sale in module form.

A more economical alternative employs the W&W EPROM Box, which, like your RAM BOX, is contained in an HP card reader case. The EPROM Box can be equipped with standard commercially available EPROM chips and has a data storage capacity of 4 - 32 kByte, in contrast to the 8 - 16 kByte capacity of a single plug-in modular EPROM.

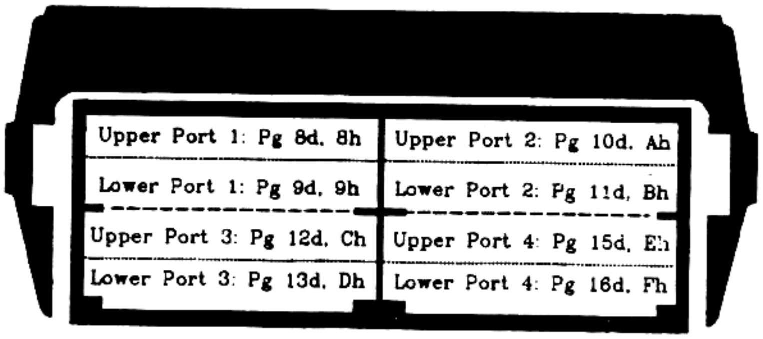
For EPROM programming services, and to develop custom User- and microcode (MCODE) software, you will always find the help you need at W&W Software.

1.3 The Organization of Read-Only Memory (ROM) in the HP-41

The HP-41 can address and thus control up to 64 kBytes of ROM, which is partitioned into 16 blocks of 4 kBytes each. Like the sections of memory in the RAM BOX, these blocks are called "pages" and are identified by hexadecimally-numbered addresses 0 - F. Specific blocks of functions in the HP-41's operating system and certain peripherals and ROMs occupy pages 0 - 7. The remaining addresses are used to communicate with application ROMs inserted in the input/output (I/O) ports of the HP-41. The addressing scheme for ROM pages is given in the Table on page 8; for page numbers above 9, addresses are given in both decimal and (in parentheses) hexadecimal notation.

ROM Page	Function
0	Reserved for the operating system (system ROM 0)
1	Reserved for the operating system (system ROM 1)
2	Reserved for the operating system (system ROM 2)
3	Reserved for the expanded operating system of the HP-41 CX (system ROM 3); not used by the HP-41C or HP-41CV
4	Reserved for the diagnostic ROM used by HP repair facilities
5	Reserved for the TIME module in the HP-41C and HP-41C. This page is used by ROMs 5a and 5b of the expanded operating system of the HP-41CX; the section currently needed is automatically switched into the active circuitry ("page switching").
6	Reserved for the ROM of the HP82143A printer and for the printer functions in the HP-IL ROM when they are enabled
7	Reserved for the HP-IL module; *note, however, that the "DISABLE" switch on the module casing readdresses the ROM to page 4, and removes the printer functions from the active circuit.*
8	Lower 4 kB section of port 1
9	Upper 4 kB section of port 1
10 (A)	Lower 4 kB section of port 2
11 (B)	Upper 4 kB section of port 2
12 (C)	Lower 4 kB section of port 3
13 (D)	Upper 4 kB section of port 3
14 (E)	Lower 4 kB section of port 4
15 (F)	Upper 4 kB section of port 4

The Figure below illustrates the correspondance between the I/O ports and the page addresses of ROM blocks.



1.4 I/O Buffers

Certain HP-41 application ROMs create and reserve for themselves a special area of main memory RAM called the I/O buffer region. The TIME module, for example, stores time alarms in this part of RAM. Situations occur when it is useful to be able to store I/O buffer contents in a secure section of memory and recall them as needed. Examples of this type of situation are:

a: The user wishes to temporarily remove a ROM from his HP-41 without losing the contents of the buffer it has created. For instance, the user is midway through a complex matrix manipulation and must unplug his CCD or Advantage ROM, but wishes to maintain the buffer containing the working matrix and element pointer so he may resume the problem upon reinserting the ROM.

b: The user requires a large number of TIME alarms but wants to have only a portion of these resident in the HP-41's main memory at any one time in order to minimize the buffer size required.

c: The user is programming his RAM BOX in microcode (MCODE) and wishes to copy the contents of the I/O buffers to a another part of memory to prevent accidental loss from, *e. g.*, a system crash.

This section of the manual presents a description of the organization of I/O buffers to help the user understand and use RAM BOX functions that create and manipulate data files for storing the contents of I/O buffers.

The term input/output (I/O) usually refers to information transfer between a computer's central processing unit (CPU) and peripherals (including, *e. g.*, the keyboard and monitor as well as printers, *etc.*). For the purpose of this discussion, however, we shall use the term in a more limited sense to denote **dialog with RAM which bypasses the operating system.** More specifically, we shall note that each application ROM (such as the TIME and CCD modules) which establishes an I/O buffer, interacts directly with its unique, specially reserved region of memory and manages this buffer independently of the operating system and all other coexisting I/O buffers.

To the operating system, an I/O buffer is an inaccessible area of RAM. The reason for this is the structure of the first register (*i. e.*, the one with the ***lowest absolute address***), which we shall call the **base register.** To the operating system, the most important data in the base register is contained in nybbles 10 - 13 (*i. e.*, bytes 5 and 6) nybbles. We can visualize the base register as containing data in the form

*iil*lxxxxxxxxxx,

in which each letter denotes one nybble. The two nybbles comprising the *ii* field each contain a copy of the recognition character unique to that buffer (*i. e.*, the buffer ID). This recognition character is a hexadecimal number from 1 to E (note that the hexadecimal digit F is used to mark key assignment registers); a Table of the ID numbers of buffers created by currently available ROMs and peripherals is given on the next page. The two nybbles in the *ll* field contain a number specifying the buffer size (number of registers).

When the HP-41 is turned on, the operating system surveys the I/O buffer region. As it locates the base register of each buffer, it clears the ID number from nybble 13 (the leftmost nybble of byte 6) and adds the number of registers given in the *ll* field to the absolute address specified by the current program pointer. The operating system then jumps to the pointer position it has just computed and tests to see if the register at that address is the base register for another buffer. When the survey of I/O buffers has been completed, the operating system polls the ROMs that are capable of establishing I/O buffers. When a ROM recognizes a buffer it has created, it rewrites the appropriate buffer identity number in nybble 13 of the base register. When this second poll of buffers is finished, the operating system executes a "PACK I/O" routine: the buffers with 0 in nybble 13 of the base register (*i. e.*, buffers belonging to ROMs which are no longer plugged into an I/O port) are cleared, and any free register space thus created is eliminated from the buffer region and moved above the ***.END.*** to program memory.

Buffer ID	ROM or Peripheral	Size (Regs)	Function
0	None	0	Cleared at turn-on
1	David ASSEM	*	Addresses of labels and program pointer
2	David ASSEM	*	Unresolved labels
5	CCD ROM	2	Word size, random number seed and active matrix
	Advantage ROM	2	Same as for CCD ROM
6	Extended IL ROM	*	Printer status
7	Sokisha ROM	1	User flags and status information
	Extended IL ROM	?	Temporary storage buffer for the multi-column program listing routine **MCPRP**
A	TIME ROM	*	Time alarms
B	Plotter ROM	^10	Bar code and plotting parameters
	Monitor	*	I/O buffer for HP-IL commands
C	HP-IL Development ROM	*	I/O buffer for HP-IL commands
D	CMT-300	^8	Status information
E	Advantage ROM	?	Temporary data storage for the functions **INTEG** and **SOLVE**
	DATAFILE and ES-41 ROMs	1	Status information for the currently active file
F	HP-41 S.Operating System	*	Key assignments

Notes:

* Indicates that the buffer is of variable size.
^N Indicates that the buffer is of variable size but is at least N registers long
"Temporary" buffers are cleared by the managing ROM upon termination of the relevant operation or function; most other I/O buffers remain intact until the first "turn-on" of the HP-41 after the managing ROM is removed.

2. GENERAL CONTROL FUNCTIONS

2.1 BUFLNG?
(Buffer Length?)

This function determines the number of registers occupied by the I/O buffer whose buffer identity number is specified in *x*. **BUFLNG?** is a vital function for the creation of files in the RAM BOX or in Extended Memory for storing the contents of I/O buffers. See the explanation of the function **CRFLBUF** (Section 4.14) for an example.

EFFECT OF BUFLNG? ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	**t**	z
z	**z**	y
y	**y**	Buffer ID
Buffer ID: 0 < x < 16	**x**	*Buffer Length*
L	**L**	L
A	**ALPHA**	A

A Note on Notation: A box like the one above accompanies each of the funcions described in this manual. The entries in the box serve two purposes: they specify the necessary format for data input required by the function and the data it outputs (both given in italics), as well as giving the locations of data in the stack registers before and after the function is executed.

2.2 KEYAS?
(Number of Key Assignment Registers?)

KEYAS? computes the number of registers in main memory RAM which are occupied by key assignment registers. One example of a practical application of the function may be found in Section 4.11, in which the function **CRFLKEY** is discussed. No input is required; the result is written to the *x*-register.

EFFECT OF KEYAS? ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	**t**	z
z	**z**	y
y	**y**	x
x	**x**	*Number of Registers*
L	**L**	L
A	**ALPHA**	A

2.3 PG?
(Page Contents?)

PG? is a programmable function which enables you to determine the contents of a specific 4 kByte block of RAM; the number of the page desired is specified in *x*. The function produces output as follows: the ROM header for the specified page is written to ALPHA, and a number of the form *xx.nn* is written to *x*, for which *xx* is the ROM ID part of the XROM number). *nn* is the total number of functions and/or programs contained on the page; note that the header is counted as a function.

EFFECT OF PG? ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	z
z	z	y
y	y	*Page Number*
Page Number	x	*(ROM ID).(Number of Functions)*
L	L	L
A	ALPHA	*ROM Header*

As an example of the use of *PG?*, we execute the keystroke sequence
8
PG?,
to poll the page containing the RAM BOX's oprating system, which has the ROM ID number 14 and contains 34 functions, yielding these data:

EFFECT OF PG? ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	Z
z	z	Y
y	y	*8*
8	x	*14.34*
L	L	L
A	ALPHA	*-RAM BOX A*

2.4 FNC?
(Description of a Function?)

FNC? provides a programmable means of obtaining descriptive information for any function on a given page of RAM of ROM. The required input is a control number of the form ***xx.nn***, for which ***xx*** is the ROM ID and ***nn*** is the number of the function in the ROM. The data obtained using **FNC?** include

- A) The funtion's XROM number converted to a two-byte value in decimal notation (in *x*),
- B) The same byte value described above, but displayed in ***y*** in hexadecimal notation as an ASCII character string,
- C) The XROM number in ***z***, with the function number incremented by 1 and finally
- D) An ASCII character string in **ALPHA**, containing
    1) The XROM number is positions 1 - 5, followed by
    2) Two spaces, then
    3) The function name, in positions 8 - 18 followed by
    4) Another two spaces, and finally
    5) In positions 21 - 24, the address of the first byte of the function.

FCN? behaves as a true HP-41 conditional; when it is executed as an instruction in a running program, the step following **FNC?** is executed if the specified function exists, and skipped if the function is nonexistent.

EFFECT OF FNC? ON THE CONTENTS OF THE STACK REGISTERS

INPUT	**OUTPUT**	
t	t	y
z	z	*xx.(nn + 1) [XROM Number + 1]*
y	y	*"BB.BB" (Fcn. Hex Byte Value)*
xx.nn (XROM Number)	x	*bbb.bbb (Fcn. Dec Byte Value)*
L	L	L
A	**ALPHA**	*"XX.NN NAMExxxxxxx Addr"*

As an example of the use of **FCN?** we present the program "PRFAT" below; it outputs to a ThinkJet printer a list of the function name, XROM number, hex and decimal byte value and address for each function on the page specified. We have selected as the example page 8 of the RAM BOX, on which the resident operating system is located.

Program Listing for "PRFAT"

```
␊12:52AM 12/15␍
␊ 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 of the Function List for RAM BOX Page 8, Using "PRFAT"

```
12/15/1986          -RAMBOX  1B

XROM#  Function      Adr.  Byte(H)  Byte(D)
30.00  -RAMBOX 1B    8058   A7.80   167.128
30.01  BUFLNG?       894C   A7.81   167.129
30.02  CLLSTFL       848E   A7.82   167.130
30.03  CLPG          8D82   A7.83   167.131
30.04  CLRFL         8ADA   A7.84   167.132
30.05  COPYPG        8111   A7.85   167.133
30.06  CRDIR         8192   A7.86   167.134
30.07  CRFLBUF       897D   A7.87   167.135
30.08  CRFLDTA       8965   A7.88   167.136
30.09  CRFLKEY       8971   A7.89   167.137
30.10  ENDPG         872C   A7.8A   167.138
30.11  FNC?          888A   A7.8B   167.139
30.12  FRBYT?        8225   A7.8C   167.140
30.13  GTBUF         8AF1   A7.8D   167.141
30.14  GTKEY         8B97   A7.8E   167.142
30.15  GTREG         8B42   A7.8F   167.143
30.16  GTREGX        8B58   A7.90   167.144
30.17  GTREGXY       8B61   A7.91   167.145
30.18  INITPG        805F   A7.92   167.146
30.19  KEYAS?        8959   A7.93   167.147
30.20  LDBUF         8A6B   A7.94   167.148
30.21  LDKEY         8A4F   A7.95   167.149
30.22  LDPGM         8270   A7.96   167.150
30.23  LDREG         8A94   A7.97   167.151
30.24  LDREGX        8AA8   A7.98   167.152
30.25  LDREGXY       8AB1   A7.99   167.153
30.26  PG?           8809   A7.9A   167.154
30.27  PGSUM         86BF   A7.9B   167.155
30.28  PTCT          8629   A7.9C   167.156
30.29  READPG        8EC2   A7.9D   167.157
30.30  SETPRV        8622   A7.9E   167.158
30.31  UNPTCT        8631   A7.9F   167.159
30.32  WRTPG         812D   A7.A0   167.160
30.33  XQ>XR         8756   A7.A1   167.161
30.34  'PRFAT        8DBB   A7.A2   167.162
```

2.5 XQ>XR
(Transform XEQ' to XROM)

XQ>XR transforms all **XEQ** instructions in the specified program to **XROM** instructions, as long as a global label corresponding to each instruction can be found either in the RAM BOX or in a ROM currently inserted in the HP-41. Note that the ALPHA label(s) specified in these instructions, as well as the program processed by the function, may reside either in main memory RAM or on ***any page of the RAM BOX*** (see section 4.1, **LDPGM**, for additional useful formation). If the program being processed resides in the RAM BOX, executing **XQ>XR** does not reduce the amount of memory occupied by the program, as the bytes eliminated by the transformation are simply replaced by nulls. Execution speed, however, will still be considerably increased relative to that observed for the untransformed program.

EFFECT OF XQ>XR ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	z	t
z	z	z
y	y	y
x	x	x
L	L	L
*Program Name**	**ALPHA**	*Program Name*

*Note that the **"Program Name"** can be ***ANY GLOBAL LABEL*** within the program.

2.6 CRDIR
(Create a Directory Entry)

The amount of memory that the HP-41 can access on an HP-IL mass storage device is normaly limited by the addressing range of the HP-41's HP-IL module to 130 kBytes. **CRDIR**, however, enables access to the entire memory space of a Hewlett-Packard model 9114 disk drive. If you wish to create a file of any type on a disk which will exceed the normally permissible limit of 130 kBytes, simply specify the required file size in ***X*** and execute **CRDIR**.

EFFECT OF CRDIR ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Number of Registers	x	*Number of Registers*
L	L	L
A	**ALPHA**	A

2.7 SETPRV
(Set Private Status for a Program)

This function confers PRIVATE status upon a program resident either in the RAM BOX or main memory without the need for an indirect process involving an external storage medium. The function has three modes of execution; which is operative in a given case depends on the location *and* history of the program to be processed, as explained below.

1. If the program is in main memory when **SETPRV** is executed, it may be transferred to the RAM BOX using **LDPGM** and then retrieved once more to main memory using the HP-41 function **COPY**; the program its PRIVATE status through the entire sequence of operations.

2. If the program is in the RAM BOX when **SETPRV** is executed, the program acquires PRIVATE status but MAY NOT be transferred to main memory using **COPY**.

3. PRIVATE status is conferred upon the program first in main memory, and then once again after the program is loaded into the RAM BOX: as in case 2 above, the program MAY NOT again be recopied into main memory RAM.

Note that if **ALPHA** is cleared when **SETPRV** is executed the function will operate on the program, either in main memory or in the RAM BOX, to which the program pointer is currently set.

EFFECT OF SETPRV ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	z	t
z	z	z
y	y	y
x	x	x
L	L	L
*Program Name**	ALPHA	*Program Name*

*Note that the **"Program Name"** can be ***ANY GLOBAL LABEL*** within the program.

3. "HOUSEKEEPING" ROUTINES

3.1 CLPG
(Clear a Page)

This is a control function that enables you to clear the entire contents of a specified 4 kByte block of memory in the RAM BOX; *i. e.*, the block upon which CLPG operates is filled with null bytes. ***NOTE that the function does not display a warning prompt before operation***; if it is executed with a valid page number specified in *x* the contents of that page are gone forever.

EFFECT OF CLPG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
A	ALPHA	A

3.2 INITPG
(Initialize a Page)

INITPG is analogous to the **FORMAT** function in the disk operating systems of many microcomputers: it first clears the 4 kByte block of RAM whose address is given in *x*, then provides it with a ROM ID number, a name (*i. e.*, a ROM ROM header) and a directory space. This prepares the block to receive programs and data. ***NOTE that*** **INITPG** ***does not display a warning prompt before operation***; if it is executed with a valid page number in *x*, the previous contents of that page are gone forever. Once a block is opened with with this function, its contents may be manipulated until **ENDPG** is executed.

EFFECT OF INITPG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
ROM ID Number	y	*ROM ID Number*
Page Number	x	*Page Number*
L	L	L
-ROM Header	ALPHA	*-ROM Header*

INITPG will accept as a valid ROM header only the leftmost eleven characters of the current **ALPHA** string, *up to the first comma.* If **ALPHA** is cleared when **INITPGT** is executed the string "-" will be entered as the ROM header.

3.3 FRBYT?
(Number of <u>Fr</u>ee <u>Byt</u>es on a Page<u>?</u>)

This function provides the number of bytes still available for storage within the specified 4 kByte block or RAM.

EFFECT OF FRBYT? ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	z
z	z	y
y	y	*Page Number*
Page Number	x	*Number of Bytes*
L	L	L
A	ALPHA	A

Example of an Application

You should perform this sample routine in order to become familiar and comfortable with **INITPG** and **FRBYT?** [bear in mind that if you work through the example literally, the contents of page 10_d (= A_h) will be lost]. Firstly, remove all application ROMs from your HP-41; then execute the instruction sequence

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

Now execute **CAT 2**; following the sequential display of the functions of the operating system of the RAM BOX, you will see the catalog entry for the RAM page you have just initialized:

"-RAM 1A".

Now execute the sequence

10
XEQ ALPHA "FRBYT?" **ALPHA**

and see the result in *x*:

4056.00.

Thus there are 4056 Bytes (670 registers) available on page 10_d for storing programs or data.

3.4 COPYPG
(Copy a Page)

COPYPG duplicates the contents of one 4 kByte block of RAM BOX memory onto another page of the RAM BOX.

EFFECT OF COPYPG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
Number of Source Page	y	*Number of Source Page*
Number of Destination Page	x	*Number of Destination Page*
L	L	L
A	ALPHA	A

3.5 WRTPG
(Write a Page)

This function writes the contents of any 4 kByte block of the RAM BOX to a peripheral mass storage device. The number of the page to be copied and the name to be given to the new file on the medium (disk or tape) are required as inputs. The newly-created file will be categorized as file type 7 (which displays as "?" in a file directory listing) and occupy 640 registers on the storage medium. ***Note that* WRTPG *is compatible with the function* SAVEROM *included in the operating system of the ERAMCO MLDL (machine language development laboratory).***

EFFECT OF WRTPG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
File Name	ALPHA	*File Name*

3.6 READPG
(Read a Page)

This function copies to the specified page of the RAM BOX a 4 kByte file which had been previously written to a mass storage medium using **WRTPG**. The required inputs are the name of the file on the external medium and the number of the RAM page on which it is to be written. *Note that* READPG *is compatible with the function* GETROM *included in the operating system of the ERAMCO MLDL (machine language development laboratory).*

EFFECT OF READTPG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
File Name	ALPHA	*File Name*

3.7 PGSUM
(Page Sum)

PGSUM computes a checksum for the specified page *x* and writes this sum to the memory location $XFFF_h$ on that page. The result of the operation is also written to **ALPHA** and displayed in sequential messages with the first message having the format

"PG:NN RR-RR",

in which NN is the page number and RR-RR the ROM revision number. When the computation is terminated, the HP-41 will display either

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

(Recall that a checksum is determined by adding the byte values of all the data and/or program instructions contained in the block, and taking the modulus, using 256 as a base.)

PGSUM therefore serves three purposes:

1. It provides a straightfortward means of determining the revision number of any ROM, as its operation is not limited to RAM BOX pages.

2. It enables the computation of the checksum for a block of RAM and the entry of that checksum in the proper memory location on that page.

3. Perhaps most importantly, **PGSUM** compares its newly computed checksum with the sum already entered for that page and then displays a message indicating the status of the page. Specifically, if the prior and newly-computed checksums agree, the message "RR-RR INTACT" is displayed, whereas the display "RR-RR BROKEN" denotes a differing checksum. This feature of **PGSUM** enables you to determine whether data in a module or on a page of RAM has been altered. In any event, the newly computed checksum is written to the page, which you can verify by re-executing **PGSUM**.

EFFECT OF PGSUM ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
A	ALPHA	*Page No./Rev. No./Status*

3.8 ENDPG
(<u>End</u> a <u>Page</u>)

ENDPG terminates the entry of data or programs to a block of RAM, computes a checksum for the block and enters the current **ALPHA** string as the ROM header (including the revision number) for that page. ***NOTE THAT YOU SHOULD ONLY EXECUTE*** **ENDPAGE** ***WHEN YOU ARE SURE THAT YOU HAVE FINISHED WORKING WITH THE CONTENTS OF THE PAGE IN QUESTION.*** Once you have closed a page with this function, further access to its contents is impossible. Any attempt to access this page with a data storage function results in the error message "PAGE CLOSED."

EFFECT OF ENDPG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
ROM Header + Revision No.	ALPHA	*ROM Header + Revision No.*

Note that the function recognizes only the four leftmost characters in the current **ALPHA** up to the first comma. If **ALPHA** is cleared when **ENDPG** is executed, the string "----" is entered as the ROM revision number.

4. *FUNCTIONS FOR FILE MANAGEMENT*

4.1 LDPGM
(Load a Progam)

LDPGM copies to the specified RAM page in the RAM the User-code program for which a LBL is given in **ALPHA**. If **ALPHA** is clear when **LDPGM** is executed, then the program copied is the one at which the program pointer is currently positioned. With this feature of the function, the user can load the main memory with several programs to be copied to the RAM BOX, then clear **ALPHA** and use CATalog 1 to position the HP-41 successively at the individual programs and copy the whole series into the RAM BOX without having to specify a global label for each execution.

EFFECT OF LDPGM ON THE CONTENTS OF THE STACK REGISTERS

INPUT		**OUTPUT**
t	z	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
*Program Name**	ALPHA	*Program Name*

*Note that the "**Program Name**" can be ***ANY GLOBAL LABEL*** within the program.

4.2 CLLSTFL
(Clear the Last File)

CLLSTFL clears the last file on the specified page, whether it is a program, or a data, buffer or key assignment file (These are described in subsequent sections of Chapter 4. In addition, the user may find it helpful to refer to the Owner's Manual for the CCD ROM and ***Extended Functions Made Easy***, by Keith Jarett, for more detailed explanations of file types in peripheral memory devices.). If the file to be cleared was previously protected with **PTCT** (Section 4.17), this protection must be removed by the function **UNPTCT** (Section 4.18) before the file can be cleared.

EFFECT OF CLLSTFL ON THE CONTENTS OF THE STACK REGISTERS

INPUT		**OUTPUT**
t	t	t
z	z	z
y	y	y
Page Number	x	*Page Number*
L	L	L
A	ALPHA	A

4.3 CRFLDTA
(Create a Data File)

This function opens a data file having the size specified on *y* on the page given in *x*. This file may be written to and read from, using functions described in Sections 4.4 - 4.10, in a manner similar to the way data files in the Extended Functions and Extended Memory Modules are used. ***Note that if you attempt to call a data file using the keystroke sequence***

XEQ ALPHA "FILE NAME" ALPHA,

nothing will happen.

EFFECT OF CRFLDTA ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
*File Size**	y	*File Size*
Page Number#	x	*Page Number*
L	L	L
File Name	ALPHA	*File Name*

*The file size is given as the number of registers it will occupy, and must be a number < 671.
#The number of the page on which the file is to be created.

4.4 LDREG
(Load Data Registers)

This function copies into the specified data file the contents of all the data registers in main memory. If number of registers to be transferred exceeds the number of free registers in the file, no data is transferred and the error message "END OF FILE"is generated.

EFFECT OF LDREG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

4.5 LDREGX
(Load Data Registers as Directed by *x*)

This function works in a manner similar to that of **LDREG** except that it copies the contents of a defined block of main memory registers. The control number in *x* is used to determine the length and location of the block to be copied.

EFFECT OF LDREGX ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
*bbb.eee**	x	*bbb.eee*
L	L	L
File Name	ALPHA	*File Name*

* ***bbb*** is the number of the first register to be copied, while
eee is the number of the last register to be copied.

Note that here and below, the "register number" refers always to the *relative address* of the data register(s) in question.

4.6 LDREGXY
(Load Data Registers as Directed by *x* and *y*)

LDREGXY permits more control over the process of copying data to the RAM BOX than does **LDREGX** (see above), in that **LDREGXY** provides for an additional input parameter to specify the beginning position in the object data file in the RAM BOX to which the data block will be copied.

EFFECT OF LDREGXY ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
*nnn**	y	*nnn*
bbb.eee	x	*bbb.eee*
L	L	L
File Name	ALPHA	*File Name*

* ***bbb*** is the number of the first register to be copied, while
eee is the number of the last register to be copied and
nnn is the number of the register in the object data file in the RAM BOX into which the first register of the block will be copied.

4.7 GTREG
(Get Data Registers)

GTREG is the inverse of **LDREG** (see Section 4.4 above); it copies the entire contents of the specified data file in the RAM BOX to data registers in main memory, beginning with R00.

EFFECT OF GTREG ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

4.8 GTREGX
(Get Data Registers as Directed by *x*)

GTREGX is the inverse of **LDREGX**; it copies the contents of a data file to the block of data registers in main memory whose location in given by the control number in ***x***.

EFFECT OF GTREGX ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
*bbb.eee**	x	*bbb.eee*
L	L	L
File Name	ALPHA	*File Name*

****bbb*** is the number of the first register in the destination block while ***eee*** is the number of the last register of the block.

4.9 GTREGXY
(Get Data Registers as Directed by *x* and *y*)

By analogy with LDREGXY, GTREGXY permits the copying of a defined block of data from the data file in the RAM BOX, whose name is given in ALPHA, to a destination tbe block whose location is defined by the control number in *x*.

EFFECT OF GTREGXY ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
*nnn**	y	*nnn*
bbb.eee	x	*bbb.eee*
L	L	L
File Name	ALPHA	*File Name*

**bbb* is the number of the first register in the destination block, while
eee is the number of the last register of the block and
nnn is the number of the first register in the data file in the RAM BOX from which data will be copied.

4.10 CLRFL
(Clear a File)

This function clears the entire contents of the RAM BOX data file named in ALPHA.

EFFECT OF CLRFL ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

4.11 CRFLKEY
(Create a Key Assignment File)

CRFLKEY opens a file in the RAM BOX to store a complete set of key assignments. Before you can create such a file, however, you must first use **KEY?** to determine how many registers are occupied by the current set of key assignments, since the file size is required as input by **CRFLKEY** to set up the file. ***Note that if you attempt to call a key assignment file using the keystroke sequence***

XEQ ALPHA "FILE NAME" ALPHA,

nothing will happen. The file type designation is KEY.

EFFECT OF CRFLKEY ON THE CONTENTS OF THE STACK REGISTERS

INPUT		**OUTPUT**
t	**t**	t
z	**z**	z
*File Size**	**y**	*File Size*
Page Number#	**x**	*Page Number*
L	**L**	L
File Name	**ALPHA**	*File Name*

***The file size is given as the number of registers it will occupy, and must be a number < 43.**
#The number of the page on which the file is to be created.

4.12 LDKEY
(Load a Key Assignment File)

This function tranfers to the specified KEY file in the RAM BOX the complete set of key assignments currently active in main memory. Note that assignments to keys of global labels from user programs are stored as part of the label and not within the key assignment registers; thus key assignments of this type are ignored by LDKEY and its reciprocal function GTKEY.

EFFECT OF LDKEY ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

Example of an Application

This example creates a KEY file on page 10_d with the name "KEY1" and stores the current set of key assignments in the newly-created file.

KEYSTROKES	FUNCTION PERFORMED
XEQ ALPHA "KEYAS?" ALPHA	Writes to *x* the number of registers that will be required for the KEY file
10	The number of the page on which the file is to be created is keyed in; this also moves to *y* the result of the previous computation
ALPHA "KEY1" ALPHA	Places in ALPHA the name of the KEY file
XEQ ALPHA "CRFLKEY" ALPHA	Creates the file
XEQ ALPHA "LDKEY" ALPHA	Transfer of the key assignments to the new file

4.13 GTKEY
(Get a Key Assignment File)

GTKEY clears the currently active key assignment set from main memory and replaces it with the assignments stored in the KEY file specified in **ALPHA**.

EFFECT OF GTKEY ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	**ALPHA**	*File Name*

4.14 CRFLBUF
(Create an I/O Buffer File)

This function opens a file in the RAM BOX to store the contents of an I/O buffer. ***Note that if you attempt to call a buffer file using the keystrokes***

XEQ ALPHA "FILE NAME" **ALPHA,**

nothing will happen. The file type designation is BUFFER.

EFFECT OF CRFLBUF ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
*File Size**	y	*File Size*
Page Number#	x	*Page Number*
L	L	L
File Name	**ALPHA**	*File Name*

*The file size is given as the number of registers it will occupy, and must be a number < 255.
#The number of the page on which the file is to be created.

4.15 LDBUF
(Load a Buffer File)

This function tranfers to the specified KEY file in the RAM BOX the contents of the I/O buffer whose identification number is given in *x*.

EFFECT OF LDBUF ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
Buffer ID	x	*Buffer ID*
L	L	L
File Name	ALPHA	*File Name*

4.16 GTBUF
(Get a Buffer File)

This function copies to the appropriate I/O buffer the contents of the specified RAM BOX BUFFER file. If an I/O buffer with the same identification number currently exists, its prior contents are cleared and replaced by the data from the BUFFER file. If no buffer with the appropriate identification code currently exists in main memory, GTBUF creates one in which to deposit the newly copied data.

EFFECT OF GTBUF ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

4.17 PTCT
(Protect a File)

PTCT confers on the named file a special status which prevents you from inadvertently writing to, overwriting or clearing the file.

EFFECT OF PTCT ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

4.18 UNPTCT
(Unprotect a File)

This function removes from a file the protected status conferred by **PTCT**. After its execution the file may be written to, overwritten or cleared.

EFFECT OF UNPTCT ON THE CONTENTS OF THE STACK REGISTERS

INPUT		OUTPUT
t	t	t
z	z	z
y	y	y
x	x	x
L	L	L
File Name	ALPHA	*File Name*

5. ERROR MESSAGES AND THEIR MEANINGS

ALPHA DISPLAY	*FUNCTION ATTEMPTED*	*ERROR WHICH ABORTED EXECUTION*
DATA ERROR	**BUFLNG?**	The specified buffer identification code was <1 or > 14.
	CRDIR	A register number <1 or > 9,999 was specified
	CRFLBUF	Specified file size was < 1 or > 255
	CRFLDTA	Specified file size was < 1 or > 670
	CRFLKEY	Specified file size was < 1 or > 42
	FNC? or **INITPG**	An XROM number < 0 or > 31 was entered
	Any function that accesses a block of RAM	A page number < 5 was entered
ALPHA DATA	Any function for which a number is expected as input	An **ALPHA** string was entered instead of a numerical quantity
NONEXISTENT	**FNC?**	No function exists having the XROM number input to *x*
	GTREG **GTREGX** **GTREGXY** **LDREG** **LDREGX** **LDREGXY**	The data registers specified do not exist
	PG?	The page addressed is empty
	PTCT **UNPTCT**	No file exists with the name specified in **ALPHA**
	SETPRV **XQ>XR**	No program exists with the name specified in **ALPHA**
	Any function that accesses a block of RAM	A page number > 15 was entered

ALPHA DISPLAY	*FUNCTION ATTEMPTED*	*ERROR WHICH ABORTED EXECUTION*
NO NAMR	Any file management function	No file name was input to **ALPHA**
PAGE CLOSED	**FRBYT?**; any function BESIDES **CLPG** and **INITPG** which performs a storage or clearing function on a RAM page	**ENDPG** has already been used to close the specified block of RAM, preventing execution of the function called
RAM	**PTCT** **UNPTCT**	The program specified resides only in main memory RAM
ROM	**LDPGM** **SETPRV**	The program specified resides in ROM
NO ACCESS	Any function that performs a storage or clearing operation on a RAM page	An attempt was made to use the function to access the page containing the operating system of the RAM BOX
DUB XROM #	**INITPG**	The XROM number specified duplicates one already in use on another page
NO RAM	Any function that performs a storage or clearing operation on a RAM page	The desired page cannot be written to
ILLEGAL CHAR	**CRFLBUF** **CRFLDTA** **CRFLKEY** **ENDPG** **INITPG**	The specified ROM header or file name contains one or more nonallowed characters; these include ASCII codes 0 - $0F_h$, 2E, 3A or any value > 66_h (decimal equivalents are 0 - 16. 46, 58, > 102)
NO HPIL	**CRDIR** **READPG** **WRTPG**	There is no HP-IL present in the HP-41 system you are using

ALPHA DISPLAY	*FUNCTION ATTEMPTED*	*ERROR WHICH ABORTED EXECUTION*
PACKING– TRY AGAIN	LDPGM	1) The program you tried to load did have its own END, or 2) The attempt to load the program failed because of insufficient free memory on the page specified
NO ALPHA LABEL	LDPGM	The program you tried to load has no global label by which it can be detected
FAT FULL	CRFLBUF CRFLDTA CRFLKEY LDPGM	The directory space on the page you addressed does not contain sufficient room to accept the desired entry
ROM FULL	CRFLBUF CRFLDTA CRFLKEY LDPGM	The page addressed does not have sufficient room to create the desired file or load the desired program
NO FILE	CLLSTFL	No files are present on the page addressed
FL PROTECTED	CLLSTFL LDBUF LDKEY LDREG LDREGX LDREGXY	The file you tried to access has protected status and may not be manipulated by these functions
DUB NAME	CRFLBUF CRFLDTA CRFLKEY	The name given for the file you tried to create duplicates the name of an already existing file
NO KEYS	LDKEYS	No key assignments currently exist
NO BUFFER	LDBUF	No buffer exists with the specified identification number

ALPHA DISPLAY	*FUNCTION ATTEMPTED*	*ERROR WHICH ABORTED EXECUTION*
FL NOT FOUND	**GTBUF** **GTKEY** **GTREG** **GTREGX** **GTREGXY** **LDBUF** **LDKEY** **LDREG** **LDREGX** **LDREGXY**	The spcified file was not found
END OF FILE	**LDBUF** **LDKEY** **LDREG** **LDREGX** **LDREGXY**	You attempted to load more registers into the specified file than there was room to accept
All HP-IL error messages	**CRDIR** **READPG** **WRTPG**	See the HP-IL Owner's Manual for a detailed eplanation of error messages

6. DESCRIPTION OF THE WARRANTY

W&W Software Products warrants that your RAM BOX will be free of manufacturing defects for six months from the date of purchase. To submit a warranty claim, complete the enclosed warranty card, have it stamped by the dealer from whom you purchased the RAM BOX, and send it to us. When this is done, you will receive prompt service. If the lithium battery fails, we shall replace it at our expense.

We shall assume no responsibility for consequential damages, or for damage resulting from improper use of the RAM BOX.

You should always maintain backup copies of the data you enter into your RAM BOX, to gaurd against the irretrievable loss of this data which could result even from very minor damage to the equipment. Under no circumstances shall we accept responsibility for the loss of data or for consequential damages resulting from data loss.

For questions about your RAM BOX or information about our other products, write

W&W Software Products/S.O.S. Company
1850 East Seventeenth Street, Suite 102
Santa Ana, California 92701

or telephone
(714) 558-1806