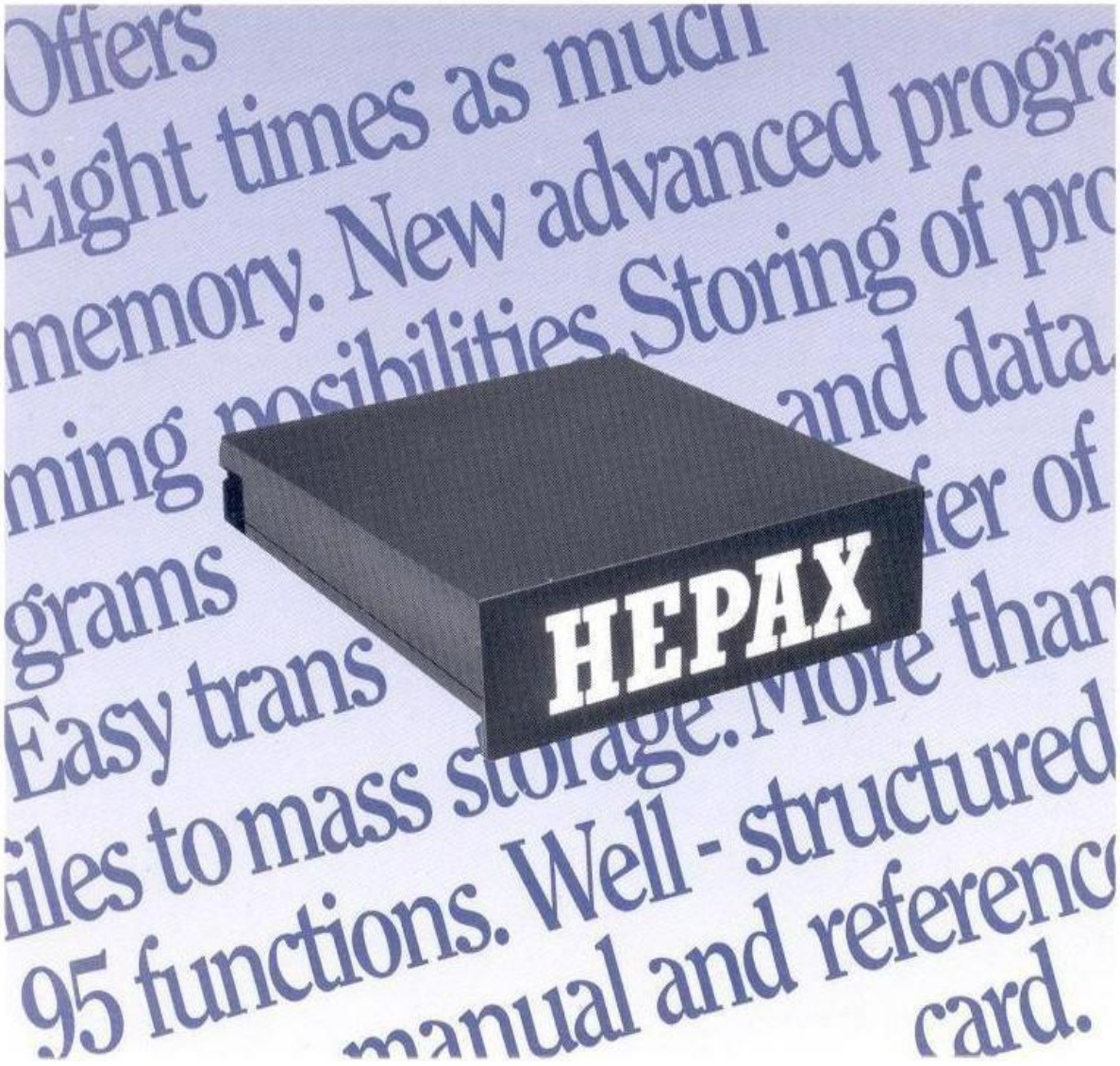
VM Electronics

HEPAX MODULE

Owner's Manual

Volume 1: Normal and Advanced Operation



The XF/XFA multi-function subfunctions

Name	Number	Function
ALENG	000	Return length of string in ALPHA.
ANUM	001	Convert string in ALPHA to numerical value in X.
AROT	002	Rotate contents of ALPHA.
ATOX	003	Convert character in ALPHA to character code in X.
CLKEYS	004	Clear all key assignments.
CLRGX	005	Clear registers as specified by X.
GETKEY	006	Get keycode depending on key pressed.
GETKEYX	007	Get keycode within time specified by X.
PASN	006	Programmable assignment.
PCLPS	009	Programmable clear programs.
POSA	010	Find position of string or character in ALPHA.
PSIZE	011	Programmable SIZE.
RCLFLAG	012	Recall the status of user flags 00-43.
REGMOVE	013	Move a block of main memory data registers.
REGSWAP	011	Swap two blocks of main memory data registers.
ΣREG?	015	Return the location of the statistical registers.
SIZE?	016	Return the current SIZE.
STOFLAG	017	Restore the status of user flags 00-43.
x<>F	018	Exchange status of user flags 0-7 with X.
XTOA	019	Convert character code in X to character in ALPHA.
X=NN?	020	Compare X with indirect Y.
X≠NN?	021	Compare X with indirect Y.
X<NN?	022	Compare X with indirect Y.
X<=NN	023	Compare X with indirect Y.
X>NN?	024	Compare X with indirect Y.
X>=NN?	025	Compare X with indirect Y.

The HEPAX multi-function subfunctions

Name	Number	Function
AND	001	Logical X AND Y.
BCAT	002	Block catalog.
BCD-BIN	003	Converts number in X from BCD to binary.
BIN-BCD	004	Converts number in X from binary to BCD.
CTRAST	005	Set display contrast ("Halfnut" calculators only).
DELETE	006	Works like DELETE of the hexadecimal editor.
INSERT	007	Works like INSERT of the hexadecimal editor.
NOT	008	Complement of X.
OR	009	Logical X OR Y.
ROTYX	010	Rotates Y register X nybbles.
SHIFTYX	011	Shift Y register X bits.
XOBR	012	Logical X exclusive-or Y.
X+Y	013	Bitwise addition.
X-$	014	Converts X register to alpha string.
Y-X	015	Bitwise subtraction.

The HEPAX Module

Volume 1
Normal and Advanced Use

May 2017

Printed in Denmark. (original)

Transcribed May 2017 by Robert Meyer.

Rev 1.0: original transcription
Rev 1.1: added HEPAX Function XROM numbers table
Rev 1.2: added/corrected items from HEPAX Manual Addendum Card (Nov 1988) and other transcription errors.
Rev 1.3: minor corrections

VM Electronics, Nyelandsvej 7, 1., DK-2000 Frederiksberg, Denmark.

Contents

Part 1: Normal Use

Part II: Advanced Use

List of figures

List of tables

Introduction

Congratulations on your new HEPAX module!

The HEPAX module – ***HE***wlett-***PA***ckard 41 e***X***pansion module – is a very powerful expansion of your HP-41 system. In effect, it lets you create your own application modules, you can use data and text files exactly like in the Extended Functions module, you get all the functions of the XF module and you can save key assignments and entire calculator memory exactly like with the HP-IL module and a mass storage device.

The manual thoroughly describes all functions and gives many examples. For those interested, it also gives much useful information about the inner structure of the HP-41 and gives a complete overview of the HP-41 assembly programming.

Terminology

In this manual, the term *main memory i*s used to describe the up to 319 directly accessible registers of the HP-41CV (the HP-41C may be expanded to this amount by using Hewlett-Packard memory modules). *Extended memory* is the memory that may be added to the HP-41 system by using the HP82180A Extended Functions/Memory module and two HP82181A Extended Memory modules. The term *HEPAX memory* is used to describe the up to 31408 words of expanded memory that may be added to the HP-41 system by using HEPAX modules

The basic unit of HEPAX memory is a *word*. Of the 8192 words in the Standard HEPAX and HEPAX Memory modules, 340 words are used internally by the HP-41 itself and the HEPAX file system – leaving 7852 words available to the User. The Advanced HEPAX and HEPAX Double Memory modules contain twice as much memory.

In order to achieve maximum storage capacity, the HEPAX file system operates with two types of registers: *HEPAX data registers* consisting of 6 words and *HEPAX program registers* consisting of 7 words. Refer to section 4: “The HEPAX file system” for a detailed explanation of this subject.

With the HEPAX module, you can program the HP-41 in machine language. This is the “native” language of the HP-41 microprocessor. HP-41 machine language is known as microcode, machine code, or simply M-code. We will use the term *M-code* throughout this manual.

In Part III and IV of this manual, we will often use numbers written in binary or hexadecimal form. Binary numbers are identified by the suffix “b” and hexadecimal numbers by the suffix “h”. Thus, 247 = F7h = 11110111b. Refer to appendix D for more information about binary and hexadecimal numbers.

CAUTION

The HP-41 must be OFF before you insert any HEPAX module!
Before plugging in your HEPAX module, make sure that you understand the following section, “Inserting and Removing HEPAX modules.”

Inserting and removing HEPAX modules

Before inserting the HEPAX module for the first time, take the time to read through this entire section.

Up to four HEPAX modules may be plugged into the HP-41 ports. This would give you a maximum of 31,408 words or 5,222 HEPAX data registers of extra memory.

To minimize power consumed by the HEPAX module, insert it in port 1. Other configurations will reduce battery life slightly. Try to avoid placement in port 4. If the module is removed from your HP-41, it will lose its contents.

CAUTION

Turn the HP-41 OFF before inserting or removing a HEPAX module! Failure to do so could damage both the calculator and the HEPAX module.

Configurations

Using the HEPAX with HP-41C memory modules

In an HP-41C, the HEPAX module must be placed in a port with a higher number than the last HP memory module (the HP-41CV and CX do not use memory modules). The port numbers are indicated on the back of the calculator.

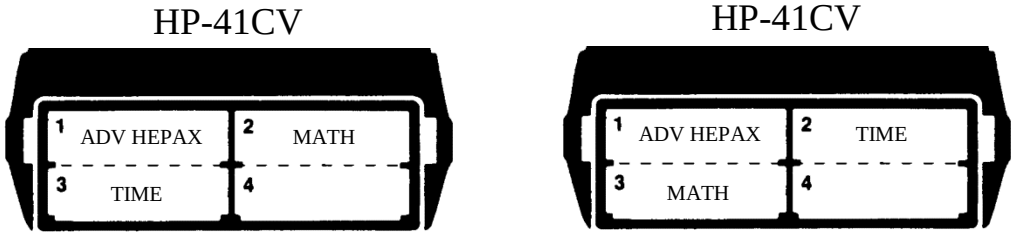
Using Advanced and Double memory modules

If you are inserting an Advanced HEPAX module or a HEPAX Double Memory module, the port next to the module must be empty or contain a system addressed device. All system addressed devices are show below.

HP-41C memory modules
Extended Memory modules
TIME module
HP-IL module
HP-82242 IR printer module
HP-82143A printer

Table 1, System addressed devices

Example:



Wrong!

(ADV HEPAX is not next to a system addressed device)

Right!

Fig. 1, Using Advanced and Double Memory HEPAX modules

Using HEPAX together with HP-IL

Four configurations with HEPAX and the HP-IL module (switch set to "enable") are illegal. If you turn on the HP-41 with an illegal configuration, you will get the message ILL CONFIG. Turn the HP-41 off and set the switch on the HP-IL module to "disable," or remove any module or peripheral.

The four illegal configurations are shown below. Note that the configurations are illegal, regardless of which port each module or peripheral is inserted in.

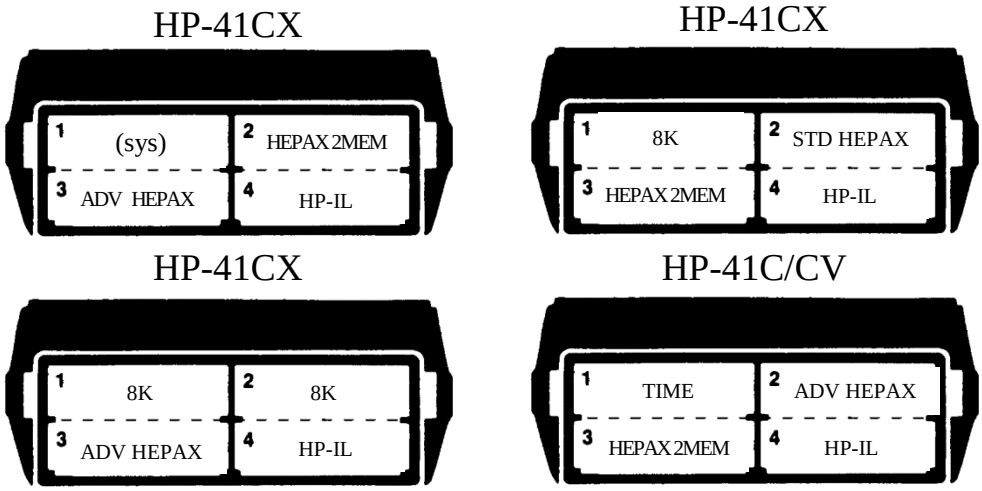


Fig. 2, Illegal configurations

"(sys)" is any system addressed device as listed in Table 1. "8K" is a module that contains 8000 words of memory or more. Most modules are known as "4K", but a few "8K" modules exist. Some of the most common 8K modules are listed in Table 2 below.

HEPAX Memory modules
HP-IL Development ROM
Plotter ROM
Advantage module
Petroleum fluids module

Table 2, "8K" modules

Ask the vendor if you are in doubt if a module is "4K" or "8K."

Identification and installation

Identification

Standard HEPAX modules are identified by the legend STD HEPAX marked on the module. Advanced HEPAX modules are identified by the legend ADV HEPAX. HEPAX memory modules are identified by HEPAX MEM and HEPAX double memory modules are marked HEPAX 2MEM.

Installing a HEPAX module

To insert a module:

Turn the calculator OFF!

Failure to do so could damage both the calculator and the HEPAX module.

Select a port to mount the HEPAX module in. Remove the port cover (save it for later use).

Insert the HEPAX module with the right side up, as shown.

Turn the HP-41 on. The HEPAX module is now ready for use.

Removing a HEPAX module

When removing HEPAX modules, some or all of the data in HEPAX memory may be lost. Modules should be removed according to the following rules:

- If modules were installed at the same time, first remove the module in the highest numbered port.
- If modules were installed at different times, first remove the module that was inserted last.

Refer to Section 4: “The HEPAX file system” for an explanation of these rules.

To remove a HEPAX module:

Turn the calculator OFF!

Failure to do so could damage both the calculator and the HEPAX module.

Put a nail under the edge at one of the lower corners of the HEPAX module and gently pull the module out as shown.

Cover the empty port with a port cover.

How to use this manual

For ease of use, this manual is divided into two volumes. This volume contains information about normal and advanced use of the HEPAX module. Volume 2 gives information about M-code programming.

Normal use

For normal use of the HEPAX module, read only Part 1 of Volume 1: "Normal Use."

If you only wish to use your HEPAX module for storing your own programs, just read Section 1: "All you need to know."

If you also want to use data and text files, or save key assignments or entire calculator memory, also read Section 2: "Other HEPAX file types."

If you wish to use all the powerful HEPAX Extended Functions (also found in the HP-82180A Extended Functions module and the HP-41CX), read Section 3: "The Extended Functions."

Advanced use

However, the HEPAX module has many other, very advanced functions. These will be helpful for synthetic programming and are essential for HP-41 M-code programming – writing your own functions.

Part II (also in Volume 1) explains about the file system and the many advanced functions of the HEPAX module. These functions include some very useful functions for handling whole blocks of HEPAX memory, a powerful disassembler, a very advanced hexadecimal editor and many others.

Part III (in Volume 2) covers the inner secrets of the HP-41 in detail giving information about the way the HP-41 works. This is important reference material in its own right, but is also a must for M-code programming.

Part IV (also in Volume 2) explains all HP-41 M-code instructions, gives detailed information about the uses of peripheral units and about creating your own ROM.

Reference information

For easy reference, this manual contains a subject index, a function index (inside back cover) and a list of multifunction numbers (inside front cover).

All messages from the HEPAX module are listed in appendix A, and a summary of the parameters needed for each function is given in appendix B. The most necessary reference tables for M-code programming are repeated in appendix C, and appendix D explains the use of binary and hexadecimal numbers.

Part I:

Normal Use

Section 1:

All you need to know

This section will explain all you need to know to transfer your own programs to HEPAX memory. First, we will explain a few general functions of the HEPAX file system and then we will explain how to copy your programs to HEPAX memory.

Programs in HEPAX memory may be executed directly. This means that they may be deleted from main memory, herby freeing an amount of main memory for other use.

The HEPAX module also lets you save data, text, key assignments and entire calculator main memory in the form of *files*. A file is a collection of data that cannot be used directly; just like a file cabinet, you must first find the information you need and take it to your desk before working on it. This will be covered in Section 2.

As the HEPAX file handling functions see it, a program in HEPAX memory is also a file. A program in HEPAX is directly executable, but it still has a file name, a header and other file information.

Using the HEPAX file system

When a file is created, it is given a *file name* of up to 7 characters. Commas are not allowed. All file names must be unique, i.e. no two files can have the same name. If you specify a file name that is already in use, you will get the **H:DUP FL** error message. HEPAX file sizes are given in registers.

All HEPAX files use 14 words at the beginning of the file for file name, file type, file size, pointer values, etc. This is called the *header*. You need not concern yourself with these headers – they are used by the file system only.

At any time, one file will be the *current file*. The current file is the file that you are presently working on. We will refer to a file either by its file name or simply as “the current file.”

Viewing the files of the HEPAX file system

HEPDIR

The HEPDIR (HEPax DIRectory) function gives a catalog of all files in the HEPAX file system. If a printer is connected, the list will be printed instead of shown in the display.

For each file is shown the name, the file type and whether the file is SECURED. When the catalog is stopped, the displayed information and the size in HEPAX registers is returned to the ALPHA register. Refer to Table 3 below for a list of the file types.

On the printer both file name, type, secured status and file size is printed.

PR	- program
DA	- data
AS	- text (also called ASCII data)
KE	- key assignments
WA	- write-all
,S	- secured

Table 3, HEPAX file types

Several other file types exist. Refer to Section 4, “The HEPAX file system” for a complete list of file types.

After the entire catalog has been shown, the number of HEPAX data registers available in the HEPAX file system is returned to the X register. This is the largest new data file you can create – i.e. HEPDIR has already taken into account the 14 words needed for the new header.

The catalog may be temporarily halted by pressing any key except ON and R/S. When the key is released, the catalog continues. You leave the catalog by pressing the R/S key. The last file shown becomes the current file. If the catalog runs to the end, the current file is not changed.

If the catalog is empty, the message **H:DIR EMPTY** is shown.

HEPDIRX	X	file number

The HEPDIRX (HEPax DIRectory by X) function is very similar to HEPDIR, but it concerns one file only. To get information about the n'th file, place n in the X register. E.g. to get information about the second file in the HEPAX catalog, place 2 in the X register and execute HEPDIRX.

If the file exists, the name of it is returned to the ALPHA register and the file type is returned to the X register as a two-character code. The file types are listed in Table 3 above. The file is made the current file.

If the n'th file does not exist, the ALPHA register is cleared, zero is returned to the X register and the current file remains the same.

n is always saved in the LASTX register.

Managing HEPAX files

HEPROOM

At any time, you may execute the HEPROOM (HEPax ROOM) function to find out how many HEPAX data registers are available in the HEPAX file system. Just like HEPDIR, this function automatically takes into account the 14 words needed for the header of a new file.

HFLSIZE	ALPHA	file name

HFLSIZE	ALPHA	(empty)

To find the size of any one file, use the HFLSIZE (Hepax FiLe SIZE) function. Place the name of the file in the ALPHA register and execute HFLSIZE. The size in HEPAX registers is returned to the X register. If the ALPHA register is empty, the size of the current file is returned.

HRCLPTA	ALPHA	file name

HRCLPTA	ALPHA	(empty)

To get the size in words of any HEPAX file that is not a data or text file, you may use the HRCLPTA (Hepax ReCalL PoinTer by Alpha) function. This function is designed for use with HEPAX text and data files, but if you enter the name of a file of another type in the ALPHA register and execute HRCLPTA, the file size in words is returned to the X register. If the ALPHA register is empty, the size of the current file is returned.

HRENAME	ALPHA	old file name,file name

If you decide on changing the name of a file in the HEPAX file system, you can use the HRENAME (Hepax RENAME file) function. Write the old file name in the ALPHA register, write a comma and the new file name, and execute the HRENAME function. If you attempt to rename a file to a name that is already in use, you will get the **H: DUP FL NAME** error message.

Example:

You have a data file name “DTA” that you wish to rename to “NUMBERS”:

Keystrokes	Display:	
XEQ HEPDIR	DTA DA	The old file name
ALPHA DTA, NUMBERS	DTA,NUMBERS_	Enter old name,comma, new name
ALPHA	0.0000	
XEQ HRENAME	0.0000	The file is renamed
XEQ HEPDIR	NUMBERS DA	The file now has the new name

HPURFL	ALPHA	file name

If you no longer need a file in HEPAX memory, use the HPURFL (Hepax PURge FiLe) function to delete it. Place the name of the file to be deleted in the ALPHA register and execute HPURFL. The remaining files are automatically packed for efficient memory usage.

Although HPURFL is programmable, it should never be used in a program in HEPAX memory.

If you have assigned a program in HEPAX memory to a key, you may sometimes find that the next program in CATALOG 2 is now assigned to that key. You should check any keys with programs in HEPAX assigned after using PURFL on a program. Refer to Section 4: “The HEPAX file system” for more information about this.

Transferring HEPAX files to and from Mass Storage

All HEPAX file types are compatible with the Mass Storage file types. This means that programs, data, text, key assignment and write-al files created using the HP-IL module may be read directly into the HEPAX module and that HEPAX files transferred to mass storage may be read using the HP-IL module functions.

Almost full text files will be slightly expanded when transferred to mass storage by HWRTFL. If the file is already maximum size, or the HEPAX module is full, this may cause problems when retrieving the file using HREADFL.

HWRTFL	ALPHA	file name
HWRTFL	ALPHA	HEPAX name,Mass storage name

To write a HEPAX file to a file in Mass Storage with the same name, enter the name of the file in the ALPHA register and execute the HWRTFL (Hepax WRiTe FiLe) function. You can also give the Mass Storage file another name by entering the HEPAX file name, followed by a comma and the Mass Storage file name, in the ALPHA register before you execute HWRTFL.

If a file of the same type already exists in Mass Storage with the given file name, the previous file is overwritten with the new file.

HREADFL	ALPHA	file name
HREADFL	ALPHA	Mass storage name, HEPAX name

To read a Mass Storage file to a HEPAX file with the same name, enter the name of the file in the ALPHA register and execute the HREADFL (Hepax READ FiLe) function. The HEPAX file may be given another file name by entering the Mass Storage name, followed by a comma and the HEPAX file name in ALPHA before you execute HREADFL.

If a HEPAX file of the same type with the given name already exists, the previous file is overwritten with the new file.

The HP-IL disc drive is fully supported by HWRTFL and HREADFL.

Securing HEPAX files

HSEC	ALPHA	file name
HSEC	ALPHA	(empty)

A file in the HEPAX file system may be secured against accidental loss using the HSEC (Hepax SECure file) function. Place the name of the file to be secured in the ALPHA register and execute HSEC. If the ALPHA register is empty, the current file is secured. A secured file cannot be deleted, renamed or changed (you will get the **H:FL SECURED** message).

HUNSEC	ALPHA	file name
HUNSEC	ALPHA	(empty)

The complementary function to the HSEC function is the HUNSEC (Hepax UNSECure file) function. Place the name of the file in the ALPHA register and execute HUNSEC to unsecure the file. If the ALPHA register is empty, the current file is unsecured.

Programs in HEPAX memory

If you need more programs than main memory allows, you may place some (or all) of them in HEPAX memory.

Programs residing in HEPAX memory are not really files – they just have some features in common with files. They are actually more like programs in application modules (like the MATH module). They appear in CATALOG 2 and can be executed directly. Just like programs in application modules, they cannot be edited (you will get the **ROM** message), but you can use the COPY function to copy them to main memory for editing.

Program names and file names

Usually, you will want to use the name of the program as the file name. To do this, simply enter the name of the program in the ALPHA register.

You may, however, specify a file name different from the program name. This file name will be shown in HEPDIR, but you will still have to use the program name to execute or assign the program. To save a program in main memory under a different file name, enter the program name, a comma, and then the file name in the ALPHA register. If you specify a file name, but no program name, the current program in main memory is saved in HEPAX under this file name.

Saving programs in HEPAX

HSAVEP	ALPHA	file name
HSAVEP	ALPHA	program name,file name
HSAVEP	ALPHA	,file name

To save a program in main memory to HEPAX memory do the following:

1. Enter the program name and/or file name in the ALPHA register.
2. Execute the HSAVEP (Hepax SAVE Program) function.

   The HP-41 will show **PACKING**, then **H: SAVING**, followed by **H:COMPILING**. The program now resides in HEPAX memory, the file name appears in HEPDIR, and the program name appears in CATALOG 2[1]. The program can be executed directly.
   In order to free an amount of main memory, simply:
3. Clear the program using the CLP function.

[1] Note that the HP-41CX only displays ROM headings in CATALOG 2. To see programs in HEPAX, stop the catalog before the HEPAX module, and press ENTER to list all functions and programs.

If the HSAVEP finds that your program contains a jump to a numerical label not in the program, it will give the message **NO LBL xx**, where xx is the label number. If you have used a short-form GTO instruction and the jump distance is too long, you will get the **GTO xx SHORT** message. These messages are for your information only.

If the program name (or file name if specified) already exists in the HEPAX file system, the previous file is overwritten. If you have assigned a program in HEPAX memory to a key, you may sometimes find that the next program in CATALOG 2 is now assigned to that key. You should check any keys with programs in HEPAX assigned after overwriting old programs in HEPAX. Refer to Section 4: "The HEPAX file system" for more information about this.

If you get the **H:FAT FULL** message, create a small data file and try again. If you still get **H:FAT FULL**, resize the file upwards until the HSAVEP function is successful.

Example:

Keystrokes:	Display:	
PRGM		Enter program mode
LBL T S T	01 LBL 'TST	Label of a test program
ALPHA P R G		
(space) I N	02 'PRG IN _	
(space) H E P A X	RG IN HEPAX _	A message
AVIEW	03 AVIEW	
ALPHA		
GTO ..	PACKING	
	00 REG 216	
PRGM	0.0000	Leave program mode.

You have now created a short test program in main memory.

Keystrokes:	Display:	
ALPHA T S T		
ALPHA	TST	Put the name of the prog in ALPHA
XEQ HSAVEP	PACKING	Save the program in HEPAX
	H:SAVING	
	H:COMPILING	
XEQ HEPDIR	TST PR	The TST program appears in the
	1298.0000	HEPAX directory. The number in the X register is the number of HEPAX data registers available.

Keystrokes:	Display:	
XEQ CLP ALPHA		
T S T ALPHA	PACKING 1298.0000	Clear the TST program from main memory.
CLX	0.0000	
XEQ TST	PRG in HEPAX	The TST program is executed from the HEPAX module!

If you now press PRGM to enter program mode, you can single-step through the TST program in HEPAX. If you try to insert new lines or delete lines, you will get the **ROM** message.

Now let's try to edit the TST program:

Keystrokes:	Display:	
GTO ..	PACKING 0.0000	
XEQ COPY ALPHA T S T ALPHA	COPY_ 0.0000	Copy the TST program back to main memory.
PRGM	01 LBL 'TST	Enter PRGM mode and see the TST program, now in main memory.
GTO .003	03 AVIEW	Go to line 3.
PSE	04 PSE	Insert new lines.
ALPHA E D I T E D (space) P R G	04 'EDITED _ EDITED PRG _	
AVIEW	06 AVIEW	
BEEP	07 BEEP	
PRGM		Leave PRGM mode.
ALPHA T S T ALPHA	 0.0000	
XEQ HSAVEP	PACKING H:PURGING H:PACKING H:SAVING H:COMPILING 0.0000	Save the new version The old version is automatically deleted from HEPAX. HEPAX memory is packed

Keystrokes:	Display:	
XEQ CLP		
ALPHA T S T		
ALPHA	PACKING 0.0000	Delete the program from main memory.
XEQ TST	PRG IN HEPAX EDITED PRG (beep)	The edited program runs in the HEPAX module.

Making programs PRIVATE

PRIVATE	ALPHA	file name

To ensure that other users cannot view or modify your programs, you can make them PRIVATE, just like with the card reader. A PRIVATE program cannot be viewed, listed, or single-stepped. It can only be executed or deleted!

To make a program in HEPAX memory PRIVATE, simply enter the name of the program in the ALPHA register and execute the PRIVATE function. If you attempt to make a secured program in HEPAX private, you will get the **H:FL SECURED** message. The program is not made private.

We look at the TST program again:

Keystrokes:	Display:	
ALPHA T S T		
ALPHA	0.0000	
XEQ PRIVATE	0.0000	The program is now PRIVATE
XEQ COPY		
ALPHA T S T		
ALPHA	PRIVATE	You can’t copy it.
XEQ TST	PRG IN HEPAX EDITED PRG (beep)	But you can still execute it
ALPHA T S T		
ALPHA	0.0000	
XEQ HPURFL	H:PACKING 0.0000	The TST program is purged from HEPAX memory.

Calling programs in HEPAX memory from main memory

If you have a program in main memory that calls any program in HEPAX memory, you should convert all XROM to XEQ instructions. This is not needed if both programs are in HEPAX memory.

The procedure for converting XROM to XEQ is:

1. Copy the program to HEPAX memory with HSAVEP.
2. Delete it from main memory with CLP.
3. Copy it back to main memory with COPY.
4. Delete it from HEPAX with HPURFL.

Section 4: “The HEPAX file system” explains the need for this conversion.

Section 2:

Other HEPAX file types

Creating data and text files

A *data file* is a collection of HEPAX data registers. Each HEPAX data register will hold one number, just like a data register in main memory. You cannot use STO and RCL directly, but you get some other advanced functions for accessing your data.

A *text file* (also called an *ASCII file*) is a collection of text lines called *records*. The size of a HEPAX text file is given in HEPAX program registers. One HEPAX program register will generally hold 7 characters of text.

When you create a file, you must specify the name and size of the file. Just like all other HEPAX files, data and text files begin with a header that is automatically added at the front of the file.

The HEPAX module contains exact equivalents to all file handling functions of the Extended Functions module and some of the CX Extended Functions.

Creating data files

HCRFLD	X	file size	ALPHA	file name

When creating HEPAX data files, all you need to decide is how many numbers you will initially need to store. This is the number of HEPAX registers you will need – the file size you specify when creating the file. The maximum size of a HEPAX data file is 5222 registers in a configuration with 32,000 words of HEPAX memory.

Enter the file name in the ALPHA register, the file size in registers in the X register and execute the HCRFLD (Hepax CReate FiLe Data) function. The file is now created and made the current file, and the pointer is set to the first register.

Creating text files

HCRFLAS	X	file size	ALPHA	file name

When creating a HEPAX text file, you should first estimate the necessary size of the file. A rough estimate is usually sufficient, since in most cases you can resize the file later using the “HRESZFL” program. Remember that text file sizes are specified in HEPAX program registers. The maximum file size is 577 registers.

An example of a HEPAX text file containing three records is shown below in register format. This file is shown on page 32 in record format. Note that at the beginning of each record one character is used to indicate the length of the record. Also note that the end-of-file mark takes up one character.

HEADER						
012	**A**		**C**	**O**	**L**	**L**
E	**C**	**T**	**I**	**O**	**N**	**007**
O	**F**		**T**	**E**	**X**	**T**
005	**L**	**I**	**N**	**E**	**S**	**<EOF>**

Fig. 3, A HEPAX text file in register format

Thus the exact space in words needed for a HEPAX text file may be determined as follows:

1. Add up the number of characters in all records.
2. Add the number of records.
3. Add one for the end-of-file mark.
4. Divide by seven and round up to the nearest whole number.

Enter the name of the file in the ALPHA register, the file size in registers in the X register and execute the HCRFLAS (Hepax CReate FiLe AScii) function to create the file. The file is now created and made the current file, and the pointer is set to the first character of the first record.

Resizing data and text files

HASROOM

At any point, you can check how many characters are left in a text file using the HASROOM (Hepax AScii file ROOM) function. The text file must be the current file. The number returned to the X register is the number of unused characters in the file. Remember that each record takes up one extra character.

“HRESZFL”	X	new size	ALPHA	file name

If you wish to change the size of a data or text file, use the HRESZFL (Hepax RESiZe FiLe) program show below. This FOCAL program must be given the file name in the ALPHA register and the new size in the X register.

When increasing the file size, there must be enough HEPAX memory available to hold both the old and the new file at the same time.

When decreasing the file size, the program will give the **H: FL SIZE ERR** if the downsizing would result in loss of data. A register in a data file is considered in use if it contains anything but zero, and a register in a text file is considered in use if it contains any part of a record or the end-of-file marker. Use a negative value for new size to resize the file regardless of previous contents.

```
01*LBL "HRESZFL"
02 STO 00
03 ABS
04 0
05 HSEKPTA
06 RDN
07 HFLSIZE
08 ASTO 01
09 ASHF
10 ASTO 02
11 SF 25
12 HASROOM
13 FC?C 25
14 GTO 01
15 CLA
16 HAPPCHR
17 RCL 00
18 X<0?
19 GTO 02
20 RDN
21 7
22 /
23 INT
24 -
25 X>Y?
26 GTO 15
27*LBL 02
28 "$"
29 RCL 00
30 HCRFLAS
31 CF 00
32 SF 25
33*LBL 03
34 CLA
35 ARCL 01
36 ARCL 02
37 HRCLPTA
38 "$,"
39 HARCLRC
40 FC? 25
41 GTO 16
42 HRCLPTA
43 XF
44 3
45 XF
46 3
47 FC? 00
48 HAPPREC
49 FS? 00
50 HAPPCHR
51 FC? 25
52 GTO 05
53 CF 00
54 FS? 17
55 SF 00
56 GTO 03
57*LBL 05
58 -1
59 XF
60 2
61 XF
62 3
63 SF 25
64 FC? 00
65 HAPPREC
66 FS? 00
67 HAPPCHR
68 FC?C 25
69 GTO 05
70 GTO 16
71*LBL 01
72 RCL 00
73 X<0?
74 GTO 07
75 SF 25
76 HSEKPT
77 FC? 25
78 GTO 07
79 0
80*LBL 08
81 HGETX
82 X≠Y?
83 GTO 15
84 FS? 25
85 GTO 08
86 HSEKPT
87*LBL 07
88 "$"
89 RCL 00
90 HCRFLD
91 SF 25
92*LBL 09
93 CLA
94 ARCL 01
95 ARCL 02
96 HRCLPTA
97 HGETX
98 FC? 25
99 GTO 16
100 "$"
101 HRCLPTA
102 RDN
103 HSAVEX
104 FS? 25
105 GTO 09
106 GTO 16
107*LBL 15
108 "H:FL SIZE ERR"
109 AVIEW
110 GTO 10
111*LBL 16
112 CLA
113 ARCL 01
114 ARCL 02
115 HPURFL
116 "$,"
117 ARCL 01
118 ARCL 02
119 HRENAME
120*LBL 10
121 CLA
122 ARCL 01
123 ARCL 02
124 0
125 HSEKPT
126 RCL 00
127 CF 00
128 END
```

Program listing of the HRESZFL program

Clearing data and text files

HCLFL	ALPHA	file name

Place the name of a file in the ALPHA register and use HCLFL (Hepax CLear FiLe) to clear a data or text file. In a data file, all register contents are set to zero and the pointer is set to the first register. In a text file, all records are deleted and the pointer is set to the first character of the first record. The file is made the current file. Note that this function does not delete the file.

Using pointers in data and text files

Data and text files are accessed at one point at a time. This point is determined by the value of the pointer. The pointer is saved in the header.

Structure of data files

As mentioned at the beginning of this section, a data file is simply a collection of registers preceded by a header. Each register has a register number, starting with 0 and continuing to the end of the file. A data file of 5 registers is shown below.

Register number	Header (pointer = 3)	
0	data	
1	data	
2	data	
3	data	← pointer
4	data	

Fig. 4, A HEPAX data file

The value of the *pointer* is stored in the header. In the above example, the pointer value is 3, pointer to register number 3 (the fourth register in the file).

Structure of text files

A text file is a collection of text lines (called records), preceded by a header. A record may contain from 1 to 254 characters. Each record has a record number, and each character in a record has a character number. The pointer in a text file is of the form **rrr.ccc**, where the integer part (**rrr**) is the *record pointer* and the fractional part (**ccc**) is the *character pointer*. The text file from page 28 is shown below in record format.

Record number	Header (pointer=001.005)											
0	A		C	O	L	L	E	C	T	I	O	N
1	O	F		T	E	X	T					
2	L	I	N	E	S							
	0	1	2	3	4	5	6	7	8	9	10	11

character number

Fig. 5, A HEPAX text file in record format

Like in data files, the pointer value is stored in the header. Above, the pointer value is 1.005, pointing to record number 1, character number 5 (the “X” in the word “TEXT”).

Pointer operations

The HEPAX module contains 4 different functions to set and recall the pointers.

HSEKPTA	X	**rrr**	ALPHA	data file

HSEKPTA	X	**rrr.ccc**	ALPHA	text file

To set the pointer in any data or text file, use the HSEKPTA (Hepax SEeK PoinTer by Alpha) function. The file is made the current file.

To set the pointer in any data file:

1. Enter the register to point to in the X register.
2. Enter the data file name in the ALPHA register.
3. Execute HSEKPTA. The named file becomes the current file.

To set the pointer in any text file:

1. Enter the record and character to point to in the X register (**rrr.ccc**)
2. Enter the text file name in the ALPHA register
3. Execute HSEKPTA. The named file becomes the current file.

If the ALPHA register is clear, the pointer is set in the current file.

HSEKPT	X	**rrr**

HSEKPT	X	**rrr.ccc**

To set the pointer in the current data or text file, you may also use the HSEKPT (Hepax SEeK PoinTer) function. This function is similar to HSEKPTA, but it always sets the pointer in the current file as directed by the contents of the X register, regardless of the contents of the ALPHA register.

HRCLPTA	ALPHA	file name

To recall the pointer from any file, use the HRCLPTA (Hepax ReCalL PoinTer by Alpha) function. Enter the file name in the ALPHA register and execute the function. The pointer of the named file is recalled to the X register and the file is made the current file. If the ALPHA register is clear, the pointer of the current file is recalled.

HRCLPT

To recall the pointer from the current file regardless of the ALPHA register contents, use the HRCLPT (Hepax ReCalL PoinTer) function. This function always recalls the pointer of the current file.

Data file operations

The file specified in all the below data file operations must be a data file, otherwise you will get the **H:FL TYPE ERR** message.

Operations on all data registers

The HSAVER and HGETR functions are used to copy between all main memory data registers and a HEPAX data file. The contents of a main memory data register is copied to/from the data file register with the same number.

HSAVER	ALPHA	data file name

HSAVER	ALPHA	(empty)

To copy all main memory data registers to a HEPAX data file, use the HSAVER (Hepax SAVE Registers) function. Enter the name of the data file in the ALPHA register or leave the ALPHA register empty to save the data in the current file. The pointer will be placed just after the last register copied. If the number of main memory data registers (the SIZE) is larger than the size of the data file, and **H:END OF FL** message occurs, no registers are copied and the pointer is not moved.

HGETR	ALPHA	data file name
HGETR	ALPHA	(empty)

To copy all registers in a HPEAX data file to main memory data registers, use the HGETR (Hepax GET Registers) function. The name of the data file must be in the ALPHA register, or the ALPHA register must be empty to copy from the current file. This function copies data until the end of the data file or until there are no more storage registers. After HGETR, the pointer is placed just past the last HEPAX data register copied.

Operations on a block of data registers

Use the HSAVERX and HGETRX functions to copy between a block of main memory data registers and a block of the same size in the current HEPAX data file. The block of main memory data registers is specified using a control number of the form **bbb.eee** in the X register, where **bbb** is the first register in the block and **eee** is the last register. The block of HEPAX data registers starts at the pointer value and has exactly the same length as the block of main memory data registers.

HSAVERX	X	**bbb.eee**
HGETRX	X	**bbb.eee**

To copy a block of main memory data registers to a HEPAX data file, use the HSAVERX (Hepax SAVE Registers by X) function, and to copy a block of HEPAX data registers to main memory, use the HGETRX (Hepax GET Registers by X) function.

For both functions enter the control number in the X register and execute the function. The pointer will be placed just after the last register copied. If the specified block is larger than the number of HEPAX data registers from the pointer to the end of the file, you will get an **H:END OF FL** message, no copying will occur and the pointer will not be moved.

Operations on the X register

HSAVEX	X	data value

HSAVERX

To save the contents of the X register at the pointer, simply execute the HSAVEX (Hepax SAVE X register) function. To retrieve a number from a HEPAX data file at the pointer to the X register, use the HGETX (Hepax GET X register) function.

The pointer is advanced to the next register in the file.

Example:

Keystrokes:	Display:	
ALPHA D T A		
ALPHA	0.0000	Enter the file name in ALPHA register
5	5_	Enter the size in the X register
XEQ HCRFLD	5.0000	A data file is created

Now we enter some numbers into main memory data registers:

2 STO 03	2.0000	
3 STO 04	3.0000	
8.3 STO 05	8.3000	
9 STO 06	9.0000	
3.006	3.006_	Control number
XEQ HSAVERX	3.0060	Save registers 3 – 6 in the current HEPAX data file at the pointer.
XEQ XFA CLRGX		Clear main memory registers 3 through 6.

The file now looks like this:

Register number	Header “DTA” (pointer = 4)	
0	2	
1	3	
2	8.3	
3	9	
4	0	← pointer

Now let’s retrieve some data from the file:

Keystrokes:	Display:	
2 HSEKPT	2.0000	Move the pointer to register number 2
HGETX	8.3000	Recall the contents of the file register at the pointer
HRCLPT	3.0000	Recall the pointer. Note that it has been incremented.
HGETX	9.0000	Recall the contents of the next file register.

Text file operations

All the text file operations below operate on the current file. The current file must be a text file, otherwise you will get the **H:FL TYPE ERR** message.

Record operations

The HEPAX module contains three functions for manipulating whole records.

HAPPREC	ALPHA	alpha data

The HAPPREC (Hepax APPend RECord) function appends the contents of the ALPHA register as a new record at the end of the file. The pointer is set just past the last character in the appended record.

HINSREC	ALPHA	alpha data

To insert a record in the middle of a text file (at the pointer), use the HINSREC (Hepax INSert RECord) function. Position the record pointer where you wish the new record to be inserted and execute HINSREC. The contents of the ALPHA register is inserted at the pointer as a new record and all the following records are pushed further down in the file. The pointer is set just past the last character in the inserted record.

HDELREC

To delete the record at the pointer, use the HDELREC (Hepax DELete RECord) function. Position the pointer at the record to be deleted and execute HDELREC (the character pointer doesn't matter). The record is deleted and the following records are pulled up. The pointer is set to the first character of the record following the deleted record.

Character operations

HAPPCHR	ALPHA	alpha characters

There are also three character functions equivalent to the above record functions. The HAPPCHR (Hepax APPend CHaRacters) function appends a number of characters at the end of the current record. Place the characters to be inserted in the ALPHA register, set the pointer to the desired record (the character pointer doesn't matter) and execute HAPPCHR. The pointer is advanced to just past the inserted text.

HINSCHR	ALPHA	alpha characters

To insert a number of characters in the middle of a record at the pointer, use the HINSCHR (Hepax INSert CHaRacters) function. Position the pointer where you wish the characters to be inserted and execute HINSCHR. The contents of the ALPHA register is inserted at the pointer. The pointer is advanced to just past the inserted text.

HDELCHR	X	no. of characters

To delete a number of characters from the middle of a record, use the HDELCHR (Hepax DELete CHaRacters) function. Position the pointer at the first character to be deleted, enter the number of characters to be deleted in the X register and execute HDELCHR. Note that this function does not delete past the end of the current record. The characters are deleted and the following characters are pulled up. The pointer is set to the same position, i.e. the first character following the deleted characters.

Searching a file

HPOSFL	ALPHA	search string

You may search a text file for the occurrence of a string of characters using the HPOSFL (Hepax POSition in FiLe) function. Make the text file the current file and set the pointer to the place where the search is to begin. Place the search string in the ALPHA register and execute HPOSFL.

If the string is found, the pointer is set to the first character of the string, and the pointer value is returned to the X register. If the string is not found, the pointer is not moved, and -1 is returned to the X register. In both cases, the ALPHA register is unchanged.

Copying text to the ALPHA register

HARCLRC

HGETREC

There are two ways to copy text into the ALPHA register, using the HARCLRC (Hepax Alpha ReCalL ReCord) and HGETREC (Hepax GET RECord) functions, respectively. Their only difference is that HGETREC clears the ALPHA register before copying the text.

To use these functions, make the text file the current file and place the pointer at the first character to be copied. Execute HARCLRC or HGETREC. The functions copy text until the end of the record, or until the ALPHA register is full. If the last character in the record is copied, flag 17 is cleared, otherwise it is set (flag 17 is used by the HP-IL module). The pointer is placed just past the last character copied.

Example:

Keystrokes:	Display:	
ALPHA T E X T	TEXT _	The file name
ALPHA	0.0000	
10	10_	The file size
XEQ HCRFLAS	10.0000	The file is created
ALPHA S O M E		
(space) T E X T	SOME TEXT _	The first line of text
ALPHA	10.0000	
XEQ HAPPREC		Append as a new record
ALPHA L I N E S	LINES _	Next line of text
ALPHA	10.0000	
XEQ HAPPREC	10.0000	Append as a new record at the end of the file.

The file now consists of two lines:

SOME TEXT
LINES

Now let's edit the file:

0 HSEKPT	0.0000	Set the pointer to start of file.
ALPHA A (space)		
C O L L E C T I		
O N	A COLLECTION_	A line to be inserted
ALPHA	0.0000	
XEQ HINSREC	0.0000	Insert as a new record
ALPHA S O M E	SOME_	A search string
ALPHA	0.0000	
XEQ HPOSFL	1.0000	The pointer value of the first character of the first (and only) occurrence of "SOME"
4 HDELCHR	4.0000	Delete 4 characters (the word "SOME")
ALPHA O F	OF _	A new string to be inserted. Note
ALPHA	4.0000	that the pointer is still at the first character of record no. 1.
XEQ HINSCHR		The new word is inserted

The file now contains the three lines:

A COLLECTION
OF TEXT
LINES

Using key assignment and “Write-all” files

With the HEPAX module, it is possible to store key assignments and/or entire calculator memory in HEPAX files for later retrieval.

Using key assignment files

HSAVEK	ALPHA	file name
HGETK	ALPHA	file name

Use the HSAVEK (Hepax SAVE Keys) function to save system key assignments and the HGETK (Hepax GET Keys) function to retrieve them. Any assignment of FOCAL programs in main memory are not affected by the HSAVEK and HGETK functions.

To save the current system key assignments, place the name of the key assignment file in the ALPHA register and execute HSAVEK. If a key assignment file with the given name already exists it will be overwritten, otherwise a new file will be created.

To retrieve a set of system key assignments from a HEPAX key assignment file, enter the name of the file in the ALPHA register and execute HGETK. Any previous system key assignments are cancelled.

Using “Write-all” files

HSAVEA	ALPHA	file name
HGETA	ALPHA	file name

Use the HSAVEA (Hepax SAVE All) function to save the contents of entire calculator main memory and the HGETA (Hepax GET All) function to restore entire calculator main memory to the saved status.

To save the contents of main memory, place the name of the “write-all” file in the ALPHA register and execute HSAVEA. If a “write-all” file with this name already exists it will be overwritten, otherwise a new file will be created.

To retrieve the entire contents of main memory from a HEPAX “write-all” file, enter the name of the file in the ALPHA register and execute HGETA. The previous contents of main memory (programs, data, key assignments, etc.) are overwritten. The system should be configured exactly like when the file was created (including all peripherals).

Section 3:

The Extended Functions

The multi-function concept

Why multi-functions?

It is the philosophy of the HEPAX modules to make the maximum amount of memory available to you. To achieve this, the HEPAX support functions are tightly packed to fit into a bit of otherwise unused memory space.

This means, however, that there can be only 64 directly accessible functions in the HEPAX module. As the module contains many more than 64 functions, we must have used some trick! And yes, we have.

What is a multi-function?

The name of the trick is multi-functions. A multi-function is one function that gives a choice of other functions. For example, the XF multi-function described in this section gives a choice of 26 other functions. The functions that are accessed via the multi-function are called sub-functions of that multi-function, or just subfunctions.

Each subfunction is identified by either its subfunction name or its subfunction number. These numbers and names are given in this section and on the inside of the front cover. To give you the choice of entering either the name or the number, each multi-function must exist in two versions. The need for two different functions arise from an unfortunate “bug” in the HP-41 system software. The name-prompting multi-function ends on an “A”.

When calling a subfunction using the normal multi-function, you are prompted for the three-digit subfunction number. When calling a subfunction using the ALPHA-version multi-function, you must enter the subfunction name. All subfunction names and numbers are listed on the inside of the front cover.

Naturally, a multi-function will give the **NONEXISTENT** error message if you specify a nonexistent subfunction name or number.

Example:

We wish to execute the RCLFLAG subfunction of the XF multi-function. We can do this in two ways:

By subfunction number:

1. Execute XF and see the prompt XF _ _ _
2. Enter 012
3. The subfunction is executed.

By subfunction name:

1. Execute XFA and see the prompt XFA _
2. Press ALPHA R C L F L A G ALPHA
3. The subfunction is executed.

Multi-functions in programs

Subfunctions in programs are always specified by number. If you specify a subfunction by name, it is automatically converted. The subfunction takes up two lines: One line for the multi-function and one line for the subfunction number. This number does not enter the stack when running the program or when single-stepping it.

You can make the subfunction number enter the stack by using GTO line number to jump to the exact line containing the subfunction number, and then use SST to execute this one line.

If you place a multi-function immediately after a test line, the multi-function number will enter the stack if the result of the test is “false.” See the example on page 46 for a way to avoid this.

The XF multi-function

The HEPAX module contains exact equivalents to all file handling functions of the Extended Function/Memory module (described in Sections 1 and 2). In addition to these, the HEPAX module also contains exact equivalents to the remaining Extended Functions, and some of the CX Extended Functions. All of these functions have been collected under the XF/XFA multi-function.

When specifying a subfunction by number, execute the XF multi-function and enter the 3-digit subfunction number at the prompt. When specifying a subfunction by name, execute XFA, press ALPHA, enter the function name, and press ALPHA again.

Data register operations

XFA	CLRGX	X	**bbb.eeeii**

To clear a number of main memory data registers, place a control number of the form **bbb.eeeii** in the X register and execute the XFA CLRGX (CLear ReGisters by X) function (XF 005). **bbb** is the first register to be cleared, **eee** is the last register to be cleared and ii is the incremental. To clear the odd registers from 10 to 20, use the control word 11.02002. If **eee** is 0, one register is cleared. If **ii** is 0, every register is cleared.

XFA	SIZE?

The XFA SIZE? function (XF 016) returns the current SIZE, i.e. the number of main memory registers allocated as data registers.

XFA	PSIZE	X	new size

The XFA PSIZE (Programmable SIZE) function (XF 011) works just like the "normal" SIZE function, but it is programmable and takes the SIZE from the X register.

The XFA SIZE? and XFA PSIZE functions work conveniently together to form a short routine you can place at the beginning of all your FOCAL programs:

```
LBL "MYPRGM"          The label of your program
XF
16                    The XFA SIZE? function
(needed SIZE)         Calculate or just enter the needed SIZE
X<=Y?                 If the SIZE is sufficient,
GTO 00                continue.
XF
11                    Otherwise use XFA PSIZE to change.
LBL 00                Program continues.
```

XFA	ΣREG?

To find the main memory data register number of the first statistical register, use the XFA ΣREG? function (XF 015).

XFA	REGMOVE	X	**sss.dddnnn**

XFA	REGSWAP	X	**sss.dddnnn**

Use the XFA REGMOVE (REGister MOVE) function (XF 013) to move a block of registers, or use the XFA REGSWAP (REGister SWAP) function (XF 014) to swap a block of registers. **sss** is the beginning of the source block, **ddd** is the beginning of the destination block and **nnn** is the number of registers in the block.

In the case of REGMOVE, the contents of the destination block is lost. In the case of REGSWAP, the contents of the destination block is swapped with the contents of the source block.

If **nnn** is 0, a block length of one register is assumed.

Example:

Register	Data
00	1.4500
01	3.6000
02	7.8000
03	9.0000
04	11.2000
05	0.5560

To swap registers 01 and 02 with 04 and 05, place a control number of 1.004002 in the X register. **sss** is 001, **ddd** is 004 and **nnn** is 002. Then execute XFA REGSWAP. The registers now contain:

Register	Data
00	1.4500
01	11.2000
02	0.5560
03	9.0000
04	3.6000
05	7.8000

Flag operations

XFA	RCLFLAG

XFA	STOFLAG	X	flag status data

XFA	STOFLAG	X	**bb.ee**	Y	flag status

The XFA RCLFLAG function (XF 012) recalls status of flags 00-43 to the X register as ALPHA data to be stored for later use. The display will be unintelligible. The status of flags 00-43 may later be restored using the XFA STOFLAG function (XF 017). If the X register contains flag status data, all flags are restored. If the Y register contains flag status data, only flags **bb** through **ee** are restored.

Since the status of these flags determine the display format etc., you may store the flag status at the beginning of a program, let the program change the display format and restore the previous display format using STOFLAG at the end of the program.

Example:

LBL “MYPRGM”	The label of your program
XF	
12	The XFA RCLFLAG function
STO nn	Store in any main memory data register
.	
.	Your program is now free to change display format,
.	trig mode, etc.
RCL nn	The flag status is recalled.
XF	
17	The XFA STOFLAG function restores display format, trig, mode, etc.

XFA	X<>F

The XFA X<>F (X exchange with Flags) function (XF 018) exchanges the status of user flags 0-7 with a decimal number from 0-255 in the X register. In effect this lets you have many sets of 8 flags stored in different main memory data registers. You can take one set out, work on it, and then store it again.

You may also use the relationship between the flag status and the number in the X register directly for binary-decimal conversions, etc. This is done by interpreting user flags 0-7 as an 8-bit binary number[2]. A flag that is clear is a binary 0, a flag that is set is a binary 1. Recall that 8 bits are the same as a byte, and that a byte may take the decimal values 0 through 255.

Therefore, each possible set/clear combination of flags 0-7 corresponds to a decimal value 0-255. Each flag has a "weight," and a corresponding number is calculated by simply adding up the "weight" of all the flags that are set. The "weight" of each flag is shown below:

Flag	7	6	5	4	3	2	1	0
"Weight"	128	64	32	16	8	4	2	1

Fig. 6, The "weight" of flags 0-7

Example:

Flags set	binary number	decimal value
3	00001000	8
0,4,6	01010001	64+16+1=81
1,4,5,6,7	11110010	128+64+32+16+2=242

[2] See appendix D for an explanation of binary and hexadecimal numbers.

User mode operations

XFA	PASN	X	keycode	ALPHA	function name
XFA	PASN	X	keycode	ALPHA	program name
XFA	PASN	X	keycode	ALPHA	(empty)

To make key assignments from a program, use the XFA PASN (Programmable ASsigNment) function (XF 008). Place the name of the function or program in the ALPHA register and the keycode in the X register. The keycodes are the same as used with the built-in ASN function and in Figure 7 below. Note that a shifted key is represented by a negative keycode.

If the ALPHA register is clear, any key assignment to the key specified is cleared.

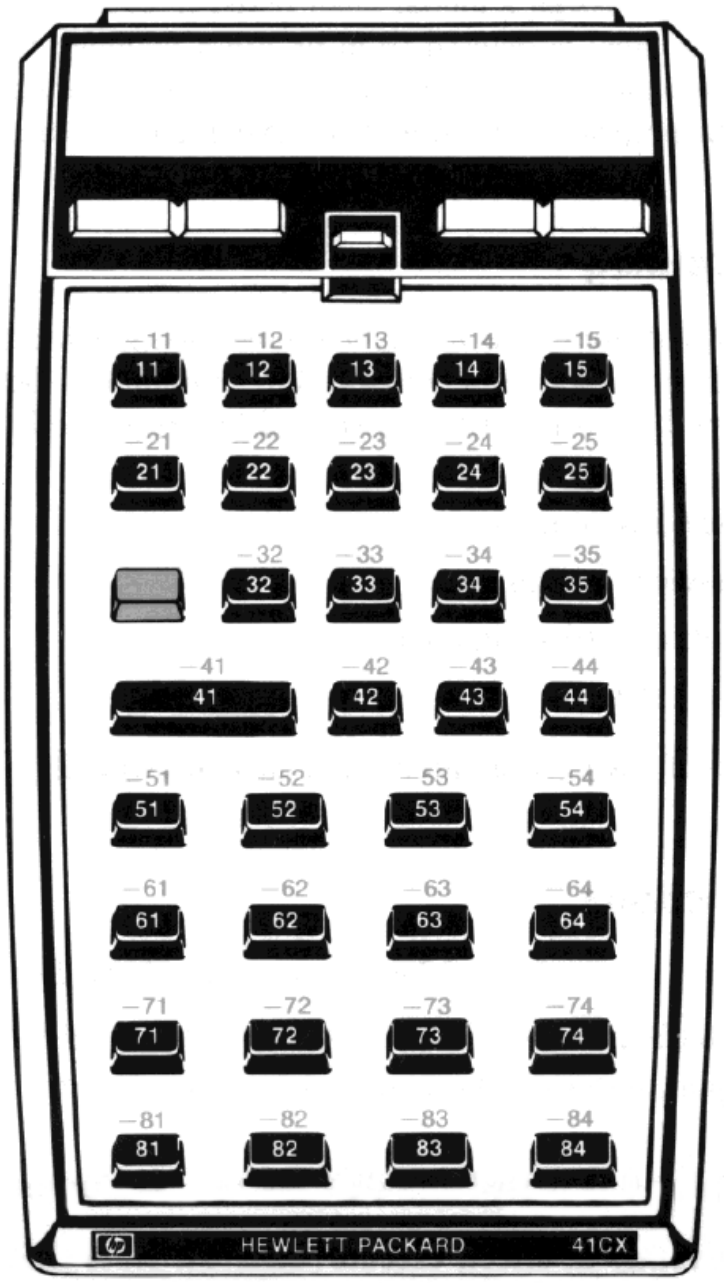


Fig. 7, User keycodes

XFA	CLKEYS

Uses the XFA CLKEYS (CLear KEY assignmentS) function (XF 004) to delete all key assignments.

ALPHA string operations

XFA	ALENG

To find the length of the string in the ALPHA register, simply execute the XFA ALENG (Alpha LENGth) function (XF 000). The length is returned to the X register.

XFA	AROT	X	no. of characters

To rotate the contents of the ALPHA register, use the XFA AROT (Alpha ROTate) function (XF 002). Place a positive number in the X register to rotate left, a negative to rotate right.

XFA	ATOX	ALPHA	character(s)

The XFA ATOX (Alpha TO X register) function (XF 003) converts the leftmost character in the ALPHA register to a character code. The characters and the corresponding code is show below in Table 4. The character is deleted from the ALPHA register and the character code is entered into the X register.

XFA	XTOA	X	character code

The XFA XTOA (X register TO Alpha) function (XF 019) is the inverse of the ATOX function. It takes a character code from the X register and appends the corresponding character to the right end of the ALPHA register.

Code	ASCII	Display	Code	ASCII	Display	Code	ASCII	Display	Code	ASCII	Display
0			32	space		64	@		96	`	
1			33	!		65	A		97	a	
2			34	"		66	B		98	b	
3			35	#		67	C		99	c	
4			36	$		68	D		100	d	
5			37	%		69	E		101	e	
6			38	&		70	F		102	f	
7			39	'		71	G		103	g	
8			40	(		72	H		104	h	
9			41	)		73	I		105	i	
10			42	*		74	J		106	j	
11			43	+		75	K		107	k	
12			44	,		76	L		108	l	
13			45	-		77	M		109	m	
14			46	.		78	N		110	n	
15			47	/		79	O		111	o	
16			48	0		80	P		112	p	
17			49	1		81	Q		113	q	
18			50	2		82	R		114	r	
19			51	3		83	S		115	s	
20			52	4		84	T		116	t	
21			53	5		85	U		117	u	
22			54	6		86	V		118	v	
23			55	7		87	W		119	w	
24			56	8		88	X		120	x	
25			57	9		89	Y		121	y	
26			58	:		90	Z		122	z	
27			59	;		91	[		123	{	
28			60	<		92	\		124	|	
29			61	=		93	]		125	}	
30			62	>		94	^		126	~	
31			63	?		95	_		127		

Table 4, Character Codes

XFA	ANUM	ALPHA	string

The XFA ANUM (Alpha data to NUMerical value) function (XF 001) converts a string in the ALPHA register to a numerical value in the X register. Note that the display format (user flags 28 and 29) should be the same as when the value was copied into the ALPHA register.

XFA	POSA	X	alpha string
XFA	POSA	X	character code

Use the XFA POSA (POSition in Alpha register) function (XF 010) to find the first (leftmost) occurrence of a character or string in the ALPHA register. The character or string to be found may be specified in the X register as either a string or a character code. Character codes are shown above in Table 4.

The position of the string or character code is returned to the X register. Positions in the ALPHA register are counted from left to right, starting with 0 at the left end. If the string is not found, -1 is returned.

Example:

You have a text string “FX=3.5643” in the ALPHA register. You decide that it would look much better if you put a parenthesis before and after the “X”:

Keystrokes	Display	
ALPHA F X = 3		
. 5 6 4 3	FX=3.5643_	
ALPHA	0.0000	
88	88_	The character code of the “X”
XEQ XFA POSA	1.0000	The X stands at position no 1.
XEQ XFA AROT	1.0000	Rotate the X to the left end.
ALPHA	X-3.5643F	The contents of the ALPHA register
ALPHA		
40	40_	The character code of “(“
XEQ XFA XTOA	40.0000	The “(“ is appended
1 XEQ XFA AROT	1.0000	Rotate the X to the right end.
ALPHA	=3.5643F(X	The contents of the ALPHA register
ALPHA		
41	41_	The character code of “)”
XEQ XFA XTOA	41.0000	The “)” is appended.
-4 XEQ XFA AROT	-4.0000	Rotate back
XEQ AVIEW	F(X)=3.5643	Looks much better, doesn’t it?
XEQ XFA ALENG	11.0000	The string is 11 characters long.
XEQ XFA ANUM	3.5643	The number in the ALPHA register.

Test functions

XFA	X=NN?	XFA	X≠NN?
XFA	X<NN?	XFA	X<=NN?
XFA	X>NN?	XFA	X>=NN?

The HEPAX module contains six tests that compare the value in the X register with a data register specified by the Y register. The contents of the Y register must be the address of an existent main memory data register (0-319, depending on SIZE), or a stack register (letters X, Y, Z, T, or L). The test functions also work on ALPHA data.

Function name	Function no.
X=NN?	020
X≠NN?	021
X<NN?	022
X<=NN?	023
X>NN?	024
X>=NN?	025

Table 5, Indirect test XF numbers

Miscellaneous operations

XFA	GETKEY

When an XFA GETKEY function (XF 006) appears in a FOCAL program, the HP-41 halts execution and waits for a key to be pressed. The keycode of the key pressed is placed in the X register. The keycodes are shown in Fig. 7 above.

If no key is pressed within approx. 10 seconds, 0 is returned to the X register and the program continues.

Note that the toggle keys at the top of the displays have keycodes 1, 2, 3, and 4, starting from the left. If you press the shift key, keycode 31 is returned to the X register.

XFA	GETKEYX	X	ss.s

When an XFA GETKEYX function (XF 007) appears in a FOCAL program, the HP-41 halts program execution and waits for a key to be pressed.

The number in the X register indicates the wait period (0-99.9 seconds).

If a numerical key is pressed, a character code is returned to the X register. The keycode (same as with GETKEY) is returned to the Y register.

XFA	PCLPS	ALPHA	program name

XFA	PCLPS	ALPHA	(empty)

Use the XFA PCLPS (Programmable CLear ProgramS) function (XF 009) to delete a program and all the following from main memory. Place the name of the first program to be deleted in the ALPHA register and execute PCLPS. If the ALPHA register is empty, the current program and all following programs are deleted.

Note that this function is programmable, as opposed to the built-in CLP function.

Part II:

Advanced Use

Section 4:

The HEPAX file system

The HEPAX file system allocates memory for the files you create. You do not need to understand the inner workings of the HEPAX file system to use the HEPAX modules – the file system is fully automatic and is normally hidden from the user.

This section is therefore not compulsory reading – but some knowledge of how the file system works might be helpful for advanced use of the HEPAX modules.

HEPAX memory

The HEPAX modules fit into the HP-41 ROM address space. Therefore, HEPAX memory is organized into words and blocks, just like HP-41 ROM memory. Refer to Section 7: "HP-41 internal structure" for a detailed description of HP-41 ROM structure.

The memory of all computers consists of a number of bits. One bit is a binary digit, and can take only the values 0 and 1. A bit is thus very much like a flag – it may be set or clear. The basic unit of HEPAX memory is a HEPAX word, consisting of 10 bits. Remember that this is different from HP-41 main memory and extended memory – the basic unit of main and extended memory is a byte, consisting of 8 bits.

4096 words together comprise a HEPAX block. In each block, 144 words are used by the HP-41 and 36 words are used by the HEPAX file system for housekeeping. This leaves 3916 words available to the user.

The HEPAX file types

The HEPAX file system uses program, data, text, key assignments and "write-all" files. All files contain a header that takes up 14 words of HEPAX memory.

Programs in HEPAX memory

Programs in HEPAX memory are directly executable copies of programs from main memory. Each program byte is copied to one HEPAX word.

The size of programs in HEPAX memory is given in HEPAX program registers – each consisting of seven HEPAX words. Thus, one register of program in main memory will take up one HEPAX program register.

Data files

A data file is a collection of HEPAX data registers. Each register will hold one number or 6 characters of text – just like a main memory data register.

Recall that one register in main memory is seven bytes = 7 x 8 bits = 56 bits. Six HEPAX words is 6 x 10 bits = 60 bits. You see that six HEPAX words is enough to store 7 bytes. To get the most from HEPAX memory, we therefore store data in HEPAX data registers that consist of only six words.

Note the difference between the size of a HEPAX program register (7 words) and a HEPAX data register (6 words).

Text files

A text file (also known as an ASCII file) is a collection of text lines. Each character in a text file takes up one word of HEPAX memory. In addition to this, one extra word is used for each text line (record) and one word is used to mark the end of the file.

Since text consists of individual characters, we must store each character in one HEPAX word. This means that we cannot pack one register (7 bytes = 7 characters) into just 6 words as with data files – we need 7 words. That’s why the size of HEPAX text files is given in seven-word HEPAX program registers.

Key assignment and “Write-all” files

Recall that HP-41 system key assignments are stored in a special part of HP-41 main memory. They always take up one register for every second assignment. Since a key assignment file is simply a copy of these registers, it can be packed just like a data file.

When saving the contents of entire calculator main memory, you save the contents of all 320 registers of main memory and the 16 status registers (the stack, the ALPHA register, etc.). These registers can also be packed like main memory data registers.

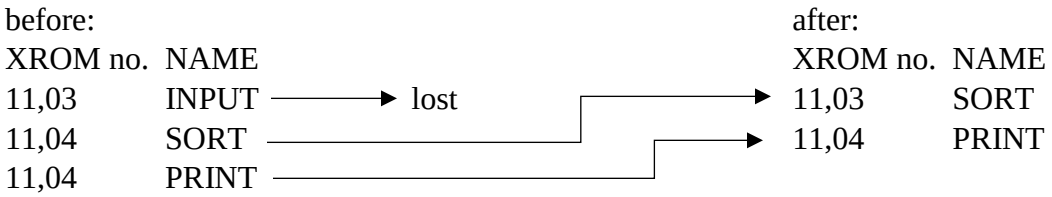
Since both key assignment and "write-all" files can be packed like data files, the size of these files is given in HEPAX data registers.

Programs in HEPAX and XROM numbers

The HP-41 uses XROM numbers to find functions and programs in peripheral units. An XROM number consists of two parts, an XROM ID no. and a function no. The HEPAX file system automatically allocates XROM ID numbers to each block of HEPAX memory, and the HP-41 numbers the functions consecutively from 00 onwards.

Because the design of the HP-41 did not foresee the development of advanced peripherals like the HEPAX module, the HP-41 may get confused when you delete a program from HEPAX memory.

Look at the below example. The HP-41 remembers that the last time it looked for the SORT program it was "Program no. 4 in external ROM no 11." We now delete the "INPUT" program from HEPAX memory:



The HP-41 is confused! Next time you press a key with "SORT" assigned, or execute an XROM "SORT" line in a program in main memory, the HP-41 looks up the fourth program in external ROM no. 11 – now the PRINT program.

Therefore, key assignments might change to the next program. Programs in main memory have the same problem – but if you convert all XROM lines to XEQ lines, the HP-41 will look for programs by name and not by number – thus overcoming the problem. The procedure for this conversion is given in the subsection "Calling programs in HEPAX memory from main memory" in Section 1.

The structure of the HEPAX file system

The HEPAX file chain

The HEPAX file system links all files together in the HEPAX file chain. The HEPDIR function displays files in the sequence of the file chain.

The files in the file chain are sorted in three groups:

1. The first group consists of all the file types that may span over several blocks (i.e. data, key assignment and “write-all” files).
2. The second group is programs in HEPAX memory.
3. The third group is HEPAX text files.

When you insert HEPAX memory, it is added to the file system the next time you turn the calculator ON. If several modules are installed at the same time, they enter the file chain with the module in the lowest numbered port first, the module in the second lowest port second, etc.

If no file system exists in any HEPAX module, you’ll get the **H:NO FILESYS** error message the first time you execute a HEPAX file system function. This may have four causes:

1. You have inserted another module or peripheral, resulting in an illegal configuration (see “Installing and removing HEPAX modules”). Remove a module or peripheral.
2. You have inserted a HEPAX module while the HP-41 was on. Press the ON key twice to turn the calculator off and back on.
3. You attempted to execute a file system function immediately after clearing entire main memory with ON/backarrow (**MEMORY LOST**). Press the ON key twice to turn the calculator off and back on.
4. You have allocated all HEPAX memory for other purposes (see below). If you want to use the file system, you must free some of the HEPAX memory allocated for other use.

If you remove a HEPAX module in the middle of the file chain, the chain will be broken at this module. This means that all files in the removed module *and* in any modules further down the file chain will be lost. Refer to the section “Installing and removing HEPAX modules” for explicit rules for removing HEPAX modules.

Never use READROM, COPYROM or CLRAM to read, copy or clear a block in the file system. Don’t write protect a block in the file system either. This would break the file chain with the above consequences.

Actual storage of HEPAX files

The actual (physical) storage of HEPAX files is somewhat different from the order of the file chain.

Within each block, files are stored in the following order:

1. Programs in HEPAX memory,
2. HEPAX text files,
3. Other file types.

There are only 64 entry points in the Function Address Table (FAT). If the current block already contains 64 programs, you will get the **H:FAT FULL** message when attempting to save the 65th program. If you create a "dummy" data file that fills up that block, the file system will automatically move on to the next block – with room for another 64 entries.

Maximum file sizes

As mentioned at the beginning of this section, HEPAX memory is divided into blocks of approx. 4000 words.

The continuous structure of program and text files means that the entire file must be stored in the same block. This gives a maximum text files size of 3916 words, or 557 HEPAX program registers. Programs in HEPAX memory will not normally be affected – main memory already limits them to 319 registers.

Data, key assignment and "write-all" files are inherently divided into registers. This enables them to be split between several blocks of HEPAX memory. The maximum size of these files is only limited by available memory – with 32000 bytes of HEPAX memory, you can create a data file of up to 5,222 registers.

Note that HEPROOM gives you the total available space in the HEPAX file system (given in HEPAX data registers). Naturally, this space may be divided between several HEPAX blocks, together adding up to the number returned by HEPROOM and HEPDIR. In the below example, there is a bit of space available in both blocks.

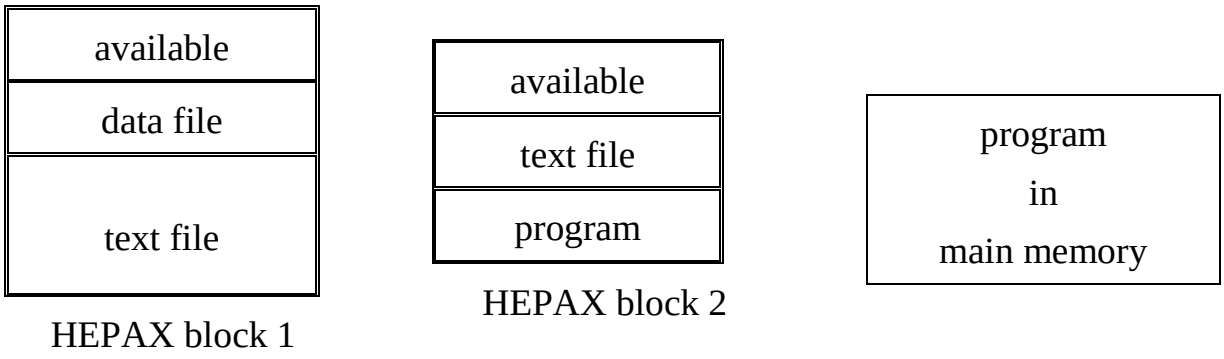


Fig. 8, Example of files in HEPAX memory

The shown program in main memory is not larger than the total available space in the HEPAX file system. It is, however, larger than any continuous space available. Therefore, it will not fit into the HEPAX file system.

Resizing files

When you use the “HRESZFL” program to resize a data or text file, what happens is that a new file of the desired length is created, the data in the old file is transferred, and the old file is deleted. Therefore, there must be enough HEPAX memory left to hold both the old and the new file at the same time.

Allocating HEPAX memory for other purposes.

Normally, you would use all available HEPAX memory for the HEPAX file system. However, it is possible to allocate whole 4K blocks of memory for other purposes.

Three words in each block determine if the block is part of the file system. To take a block out of the file system, use the HEXEDIT function to change the word at address xFF3 to 300h, and write 000h at addresses xFE7 and xFE8. To put a block back in the file system, use CLRAM to clear the block and press the ON key twice to turn the calculator off and back on.

Each time you have altered the amount of memory allocated for other purposes, you should use the HEPDIR function to restore the HEPAX file chain.

Note that any block(s) used for other purposes should *always* be the last block(s) of HEPAX memory (with the highest address).

If you insert more HEPAX memory at higher addresses, move the contents of the block(s) used for other purposes to the new, last address.

Example:

You have a Standard HEPAX module in port 2 (HEPAX memory in blocks 8 and 9) and want to use one block for other purposes. Use block 9.

You now insert a HEPAX memory module in port 4. You now have HEPAX memory in blocks 8, 9, E and F. You must now:

1. Copy the contents of block 9 to block F:
   9 ENTER 15 COPYROM
2. Clear block 9:
   9 ALPHA O K ALPHA CLRAM
3. Turn the calculator off and back on.

Section 5:

The Advanced functions

The HEPAX module contains a large number of advanced functions for handling ROM images, coding and decoding and for M-code programming.

Handling ROM images

Recall that HEPAX memory is divided into blocks of 4096 words. Each block equals one normal application module (like the MATH or STAT modules). The application modules are ROM (Read-Only Memory), and because a block of HEPAX memory is very similar to these ROMs we will refer to a block of HEPAX memory as a ROM image.

Transferring ROM images to and from mass storage

Place the file name of the ROM image(s) in the ALPHA register and a control number of the form **bb.ee** in the X register. **bb** is the first block address to be read/written and **ee** is the last block. If **ee** is zero, one block is transferred. (*Note: **bb** and **ee** are in decimal*).

Execute WRTROM to write a number of ROM images to mass storage or READROM to read a number of ROM images from mass storage. If you get a **CHKSUM ERR** message, this means that the data on the mass storage medium has been disrupted. You should check the contents of the retrieved blocks carefully.

If the ALPHA register is empty, the current position on the mass storage media is used. In this way it is possible to write many ROM images into one file and to find one ROM image in a larger file.

Don’t use the READROM or WRTROM functions to read or write blocks that are used by the HEPAX file system. Files under the file system should be transferred using HREADFL and HWRTFL.

The HP-IL disc drive is fully supported by WRTROM and READROM.

Write protecting a ROM image

To toggle the write protection status of a ROM image, use the RAMTOG function. Place the block address in the X register and execute RAMTOG. If write protection was on, it is turned off and vice versa. You are informed of the new status of the block.

Never write protect a block that is part of the HEPAX file system.

The following messages may occur when using RAMTOG:

x:WRT PRTCTED	block x is write protected.
x:NOT PRTCTED	block x is not write protected.
x:NOT RAM	block x is not RAM.
x:RAM ERROR	block x is RAM but cannot be write protected.
NONEXISTENT	a block address above 15 has been specified.

Copying and clearing ROM images

Use the COPYROM function to copy a whole 4K block of system memory to a ROM image in HEPAX memory. Place the address of the block to be copied from in the Y register and the address of the ROM image to be copied to in the X register. Then execute COPYROM.

The following messages may occur when block copying:

DATA ERROR	You attempted to copy to block 0.
NONEXISTENT	You tried to copy to or from block 16 or above.

Use the CLRAM (CLear RAM) function to clear a whole ROM image. Place “OK” in the ALPHA register, enter the block address of the ROM image in decimal in the X register, and execute CLRAM.

The following messages may occur when block copying:

DATA ERROR	The text “OK” is not in the ALPHA register, or you attempted to copy to block 0.
NONEXISTENT	You tried to clear block 16 or above.

The Disassembler

The DISASM (DISASeMble m-code) function disassembles HP-41 M-code. It may be invoked either from the keyboard or from a program. It must be supplied with a begin and an end address.

If the disassembler is invoked from the keyboard, it will prompt for start and stop addresses. You may press ENTER↑ to take the addresses from the L register as a control word of the form **000000eeeebbbb** (**bbbb** is beginning, **eeee** is end). The control word is a hexadecimal number, and may be coded with the CODE function. If the disassembler is invoked from a program, both addresses are also taken from the L register.

If a printer is connected, the disassembling is printed instead of displayed. Press R/S to stop disassembling at any point.

If you don't have a printer, you will probably want to use the DISSST program shown below to disassemble HP-41 M-code one line at a time. Enter the start and stop addresses at the prompts and see the first instruction. Press R/S to see the next line.

Program listing:

```
01 LBL "DISSST"
02 "BEGIN: "                 Enter start address
03 4
04 HPROMPT
05 "END: "                   Enter stop address
06 4
07 HPROMPT
08 CLA
09 4
10 DECODYX
11 RDN
12 LASTX
13 DECODYX
14 CODE
15 SIGN
16 SF 01
17 LBL 01
18 DISASM                    Disassemble one line
19 STOP                      Stop to view the line
20 SF 04
21 GTO 01
22 END
```

The HEPAX ROM cannot be disassembled (you will get the **HEPAX ROM** message).

The disassembling depends on the status of flags 0-4. If flag 1 is set, only one address is disassembled at a time. If flag 0 is set, the data is displayed in special formats, depending on flag 2 and 3.

Flag 2	Flag 3	Display format
clear	clear	Hex only
clear	set	Display ROM data
set	clear	ASCII data
set	set	Hex only

Table 6, Output formats of the disassembler

The second word of a LDI S&X instruction is also disassembled according to flags 2 and 3 (if both are clear, the data is shown in decimal).

If flag 4 is set, the disassembler does not look back before it starts disassembling.

The Hexadecimal editor

This function enables you to edit HEPAX memory word by word. The hexadecimal editor redefines the keyboard as shown in figure 9 below.

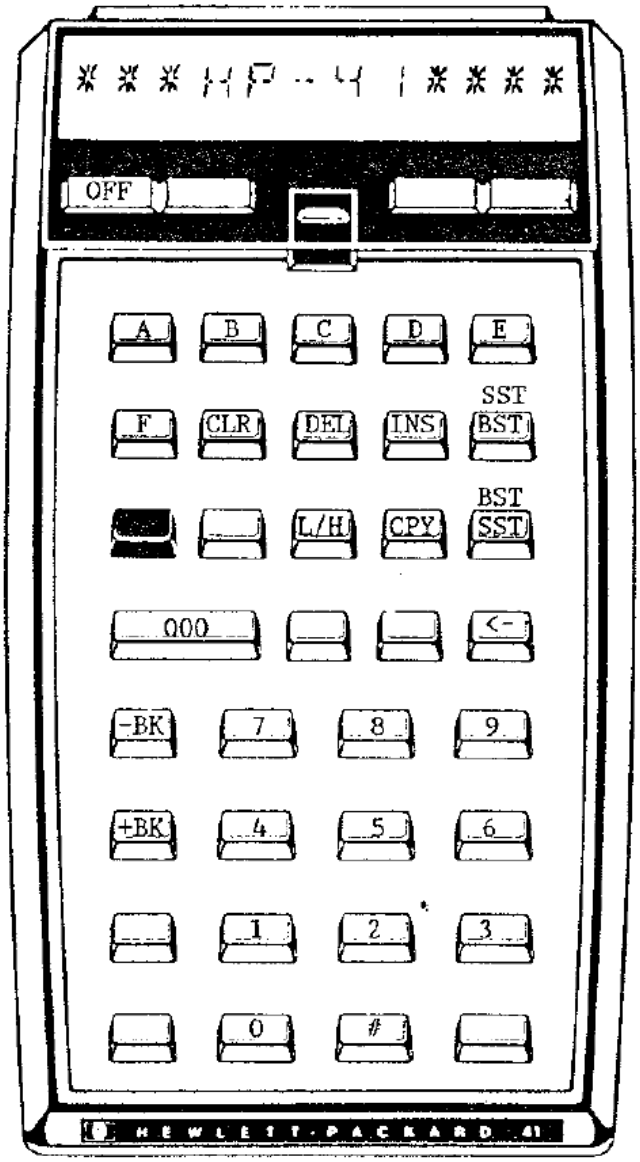


Fig. 9, The HEXEDIT keyboard

When invoked from the keyboard, HEXEDIT prompts for a start address. Press ENTER↑ to take the address from the rightmost 4 nybbles of the X register. When invoked from a program, the start address is always taken from the X register in this form.

The display looks as follows:

xabc def _ _ _ where **x** is the block, **abc** is the address, and **def** is the current contents. If the block contains several banks, the flag annunciators show which bank you are in.

Simple editing

At the displayed address, you may enter a new code using the numeric keys and/or A-F. If you wish to enter the code 000h, simply press ENTER↑.

To move one address forward, press SST. To move one address backwards, press BST (the TAN key). Pressing SHIFT causes the function of these two keys to be reversed. Note that SHIFT is in effect until you press the SHIFT key again.

To go to another address, press the backarrow once. This returns you to the address prompt. If you press the backarrow again, you leave the hexadecimal editor. If the editor was invoked from a FOCAL program, the program will continue. if SHIFT was on when you left the editor, a running FOCAL program will be stopped.

Clearing HEPAX memory

As described above, you may use the ENTER↑ key to clear the contents of the current address.

You can also use the CLR key. This key clears the contents of HEPAX memory from the current address and forwards or backwards. You are prompted for an address in the current block to clear to.

You may also clear a number of addresses from the current address and forward. Press CLR and then press the decimal point. This changes the display to #_ _ _. Enter the number of addresses to be cleared in hexadecimal.

Inserting and deleting

To delete a number of words and pull the remaining code up, use the DEL (DELete) key. When you press DEL you are prompted for an address in the current block to be the last to be deleted and then for the LIMIT – the last address to be pulled into the deleted area.

Just like with CLR, you may specify a number of addresses from the current address and forward. Press the decimal point when prompted for the last address to be deleted. The prompt changes to #_ _ _. Enter the number of addresses to be deleted.

The complementary of DEL is the INS (INSert) key. Use INS to insert NOPs at the current address and forward or backward, and push the contents of the next addresses further down or up. You are prompted for an address in the current block to be the last to be inserted and then for the LIMIT – the last address that is changed (pushed down or pulled up).

Like with DEL, you can specify a number of NOPs to be inserted. When prompted for the last address to be deleted, press the decimal point and enter the number of NOPs to be inserted.

Copying code

Use the CPY key to copy from the current address and forwards or backwards to another area. You are prompted for an address in the current block to be the last to be copied and then for the address to copy to.

The current block is suggested – press backarrow once if you need to enter an address in another block. Note that you cannot copy code to an area that crosses a block boundary.

You may specify a number of addresses to be copied like with INS and DEL above.

Special functions

HEXEDIT normally works in low-power mode to preserve power. This means that the CPU stops when waiting for input from the keyboard.

This could cause problems when working with the interrupt locations. In this case, you cannot allow the CPU to sample the interrupt locations until you are done. For this reason, the editor contains a special high-power mode that keeps the CPU running within the HEXEDIT function while waiting for input. Press L/H to toggle between power modes. Annunciator 0 will be on when you are in high-power mode.

If you are working with a bankswitched RAM device, you may press -BK or +BK to change to the next bank. If there are several banks, the display annunciators will show which bank you are in.

Messages from HEXEDIT

If you hold a key for too long, it will be **NULL**ed. The message **HEPAX ROM** means that you are attempting to edit the HEPAX ROM. If you attempt to copy, insert, delete etc. across block boundaries, you will get **DATA ERROR**.

Example:

This example assumes that you have HEPAX memory in blocks E and F. We start by taking block F out of the file system.

Keystrokes:	Display:	
XEQ HEXEDIT	ADR: _ _ _ _	
FFF3	FFF3 100 _ _ _	HEXEDIT prompts you to change the contents
300	FFF4 000 _ _ _	Address FFF3 is changed
<-	ADR: _ _ _ _	Go to another address
FFE7	FFE7 00E _ _ _	Clear the words at addresses
000	FFE8 000 _ _ _	FFE7 and FFE8
000	FFE9 091 _ _ _	
<-		Go to another address
F100	F100 3B9 _ _ _	
CLEAR	CLEAR->F _ _ _	CLEAR is the RDN key.
200	F100 000 _ _ _	F100 through F200 is cleard
012	F101 000 _ _ _	Enter some code.
123	F102 000 _ _ _	
234	F103 000 _ _ _	
345	F104 000 _ _ _	

The HEPAX memory now contains:

Address	Code
F100	012
F101	123
F102	234
F103	345
F104	000

Now we'll copy, insert and delete some code:

Keystrokes:	Display:	
BST BST		
BST BST	F100 012 _ _ _	BST is the TAN key.
COPY	COPY->F _ _ _	COPY is the RCL key.
.	COPY-># _ _ _	Copy a number of words.
004	TO: F _ _ _	Copy 4 words
104	F100 012 _ _ _	to address F104.

The HEPAX memory now contains:

Address	Code
F100	012
F101	123
F102	234
F103	345
F104	012
F105	123
F106	234
F107	345
F108	000

Keystrokes:	Display:	
SST SST		
SST SST	F104 012 _ _ _	SST is the SST key.
INS	INSERT->F _ _ _	INS is the COS key.
.	INSERT-># _ _ _	Insert a number of words.
001	LIMIT: F _ _ _	Insert one word and move code
107	F104 000 _ _ _	down until address F107.

The HEPAX memory now contains:

Address	Code	
F100	012	
F101	123	
F102	234	
F103	345	
F104	000	
F105	012	
F106	123	
F107	234	Since F107 was the limit, word at
F108	000	F108 could not be moved. Therefore, the previous word at address F107 was lost.

Keystrokes:	Display:	
BST	F103 345 _ _ _	BST is the TAN key.
DEL	DELETE->F _ _ _	DEL is the SIN key.
104	LIMIT: F _ _ _	Delete down to the address F104
106	F103 012 _ _ _	and move code up until F106.

The HEPAX memory now contains:

Address	Code	
F100	012	
F101	123	
F102	234	
F103	012	
F104	123	
F105	000	
F106	000	
F107	234	Since F106 was the limit, the words
F108	000	at F107 and down were not moved.

Copying and clearing parts of ROM images

You may also use the COPYROM function to copy a part of any system memory block to HEPAX memory.

Place a hexadecimal control word of the form **00xbbbxeeeyddd** in the X register (use the CODE function). **x** is the block to copy from, **bbb** is the first address to be copied, **eee** is the last address to be copied, **y** is the block to copy to and **ddd** is the address to copy to. Then execute the COPYROM function.

The following message may occur:

DATA ERROR	You tried to copy from more than one block in one operation, **bbb** > **eee** or you tried to copy to block 0.

The CLRAM (CLear RAM) function may also be used on a part of a ROM image, i.e. to clear any part of HEPAX memory.

Place a control word of the form **000000xbbbxeee** in the X register (use the CODE function). **x** is the block to delete in, **bbb** is the first address to be cleared and **eee** is the last address to be cleared. Then execute the CLRAM function.

The following message may occur:

DATA ERROR	You tried to clear past a block boundary, **bbb** > **eee** or you tried to clear in block 0.

Coding and decoding

Coding

The CODE function allows you to enter data in hexadecimal into all 7 bytes of the X register. This function takes the last 14 characters of the ALPHA register as hexadecimal digits and places the code in the X register. If the ALPHA register contains less than 14 characters, the number in the X register is right aligned.

Use this function to create the control words needed for COPYROM, CLRAM, etc.

For example, if you need to copy the contents of system memory addresses 8200 through 84FF to addresses 9400 through 96FF, you need the control word **00820084FF9400h** in the X register for the COPYROM function. Simply press ALPHA 8 2 0 0 8 4 F F 9 4 0 0 ALPHA XEQ CODE XEQ COPYROM.

Decoding

DECODE is the complementary of CODE. Use DECODE to decode the contents of the X register and place the corresponding 14-character string in the ALPHA register. If DECODE is executed from the keyboard, the ALPHA string is also AVIEWed. If a printer is connected, the ALPHA string is printed.

Use DECODYX to decode Y by X nybbles (one nybble = 4 bits = 1 hex digit). Takes the rightmost X nybbles of the Y register and appends them to the ALPHA register as characters. If the number in the X register is greater than 14, **DATA ERROR** is displayed.

In the above example, the X register contains **00820084FF9400h** (the display will be unintelligible). Now, if you want to know the address you are copying to, use DECODYX to decode the four rightmost nybbles: 4 XEQ DECODYX and see “9400” appended to the previous contents of the ALPHA register.

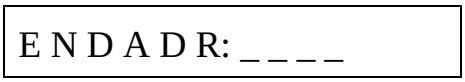
Hexadecimal prompting

The HPROMPT (Hexadecimal PROMPT) function can be very useful in your own programs. It prompts for a specific number of hexadecimal digits and allows the contents of the ALPHA register to be shown at the same time. Only as much of the ALPHA register as the number of digits allow is shown (the rightmost part). Only hexadecimal digits can be entered. The hexadecimal number is coded and returned to the X register.

Place the number of hexadecimal digits to be entered in the X register and a prompt string in the ALPHA register, then execute HPROMPT. Enter the hexadecimal number and press R/S. If you press shift before R/S, any running FOCAL program will be stopped.

Example:

The ALPHA register contains the text “ENDADR: “ and the X register contains the number 4. When you execute HPROMPT, the prompt would look like this.



Section 6:

The HEPAX multi-function

The second multi-function of the HEPAX module is the HEPAX multifunction. The subfunctions of HEPAX are various support functions that are not needed for normal use of the HEPAX module.

When executing the HEPAX function you are prompted for a 3-digit number, when executing HEPAXA, press ALPHA “function name” ALPHA. All subfunction names and numbers are listed on the inside of the front cover.

Advanced file system functions

The BCAT (Block CATalog, HEPAX 002) function lists the contents of each ROM block from block 3 to block F (HP-41CX) or block 5 to F (HP-41C/CV).

Binary functions

The NOT (HEPAX 008) function replaces X with its complement.

The AND (HEPAX 001) function performs a logical X **and** Y and places the result in the X register.

The OR (HEPAX 009) function performs a logical X **or** Y and places the result in the X register.

The XOR (HEPAX 012) function performs a logical X **exclusive-or** Y and places the result in the X register.

The BCD-BIN (HEPAX 003) function converts a decimal value in the X register to binary (right aligned). The BIN-BCD (HEPAX 004) function converts the hexadecimal code of the 4 rightmost digits in the X register to the corresponding decimal number.

The ROTYX (HEPAX 010) function rotates the Y register by X nybbles. Positive X means right, negative left. The SHIFTYX (HEPAX011) function shifts the Y register by X bit(s). Positive X means right, negative left.

The X+Y (HEPAX 013) function performs a bitwise addition of the X and Y register and places the result in the X register. The Y-X (HEPAX 015) function subtracts the contents of the X register from the contents of the Y register and places the result in the X register.

Miscellaneous functions

The CTRAST (display ConTRAST, HEPAX 005) function sets the display contrast on the newer HP-41s (the Halfnuts, known by a black rim on the display). Place a number from 0 to 15 in the X register to set the contrast. 5 is the default. This function has no effect on older calculators.

The DELETE (HEPAX 006) and INSERT (HEPAX 007) functions work just like the DEL and INS keys of the hexadecimal editor. They both need a control word of the form **00bbbbeeeellll** in the X register. DELETE deletes from address **bbbb** to **eeee** and pulls the remaining code (until **llll**) up. INSERT inserts NOPs from address **bbbb** to **eeee** and pushes the remaining code down until **llll**.

The X-$ (X register to text, HEPAX 014) function converts the contents of the X register to an alpha string, losing the first nybble. This allows you to store all kinds of data in main memory data registers, without the “normalization” that normally occurs when the HP-41 tries to make a number out of your data.

Subject index

HEPAX function XROM numbers

Function	XROM #
HAPPCHR	7,01
HAPPREC	7,02
HARCLRC	7,03
HASROOM	7,04
HCLFL	7,05
HCRFLAS	7,06
HCRFLD	7,07
HDELCHR	7,08
HDELREC	7,09
HEPDIR	7,10
HEPDIRX	7,11
HEPROOM	7,12
HFLSIZE	7,13
HGETA	7,14
HGETK	7,15
HGETR	7,16
HGETREC	7,17
HGETRX	7,18
HGETX	7,19
HINSCHR	7,20
HINSREC	7,21
HPOSFL	7,22
HPURFL	7,23
HRCLPT	7,24
HRCLPTA	7,25
HREADFL	7,26
HRENAME	7,27

Function	XROM #
HSAVEA	7,28
HSAVEK	7,29
HSAVEP	7,30
HSAVER	7,31
HSAVERX	7,32
HSAVEX	7,33
HSEC	7,34
HSEKPT	7,35
HSEKPTA	7,36
HUNSEC	7,37
HWRTFL	7,38
PRIVATE	7,39
CLRAM	7,40
CODE	7,41
COPYROM	7,42
DECODE	7,43
DECODYX	7,44
DISASM	7,45
HEPAX	7,46
HEPAXA	7,47
HEXEDIT	7,48
HPROMPT	7,49
RAMTOG	7,50
READROM	7,51
WRTROM	7,52
XF	7,53
XFA	7,54

Function Index