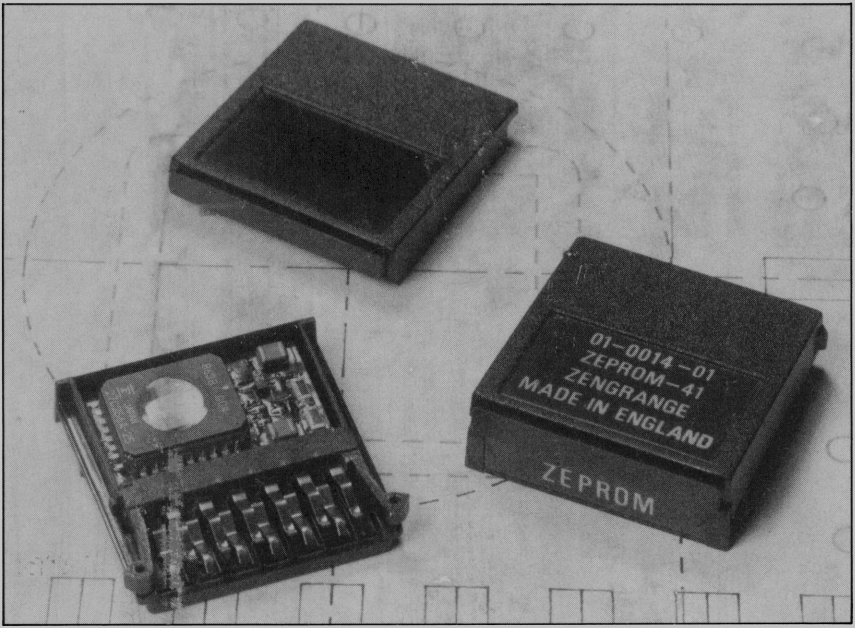
zeprom

PROGRAMMER'S MANUAL



An Erasable Programmable Read Only Memory Module

for the HP-41 Handheld Computer

zengrange

ZEPROM Module

An Erasable Programmable Read Only Memory Module
for the HP-41 Handheld Computer

Programmer's Manual

April 1988

Publisher

Zengrange Ltd, Greenfield Road, Leeds, LS9 8DB, England.

Printing History

Edition 1 ....2 April, 1988

Copyright

Trademarks

The following names are trademarks of their respective companies:

ZEPROM & ZENROM	Zengrange Limited, Leeds, England.
Zengrange	Zengrange Limited, Leeds, England.
HP-41, HP & Hewlett-Packard	Hewlett-Packard Company, U.S.A.

Notice

No expressed nor implied warranty is made with regard to this manual (including any information, examples, keystrokes, procedures or program material that it may contain) nor to its merchantability or fitness for any particular purpose. This manual is made available solely on an "as is" basis, and the entire risk as to quality and performance is with the user. Should any of this material prove defective, the user (and neither the producer nor any other party) shall bear the entire cost of all necessary correction and incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of this material.

2 April, 1988 at 2:01

Contents

Introduction

The ZEPROM Module and its programming fixture, the ZEPROM Voltage Converter, continue the Zengrange commitment to providing innovative, user oriented solutions for Hewlett Packard's hand-held and portable computers. This philosophy began with the ZENROM Programmer's Module; which for the first time opened up HP-41 Synthetic and Machine Code programming (M-code) to the non-technical user without requiring complex computer hardware or software.

The ZEPROM module continues that tradition by providing users with an ability to store favourite programs, utility routines and data in a plug-in module. In this way they are free from accidental erasure, corruption and tampering and are always available to the HP-41 without the need to load from card, tape or disc. In the past, burning a module entailed either buying expensive programming fixtures, or ordering a minimum of 100 ROM modules from Hewlett Packard. With its two companion products, the ZEPROM Voltage Converter and the PROGRAMMER ROM software, ZEPROM can even be burnt from the HP-41 itself, at a reasonable cost and by the non-technical user.

The ZEPROM module has been designed to be software compatible with the custom and application ROM modules produced by Hewlett Packard, thereby allowing ZEPROM to be used for debugging code prior to committing to ROM manufacture. Compatibility even extends to the ability to operate ZEPROM in a bank-switching mode.

To ensure durability and reliability under the most testing of conditions, ZEPROM has been designed to the exacting standards of the British Ministry of Defence and has been component and type tested to MIL STD 883C and BS9400.

About this Manual

This manual is designed for users intending to program a ZEPROM module. It therefore assumes that the user is knowledgeable about HP-41 user code programming and has a good general understanding of the HP-41. The information provided, used together with the utility routines pre-programmed into one 4K core of ZEPROM, will easily enable a user to burn user code programs. Where different requirements exist, a user can quickly write user code programs using the PROGRAMMER ROM functions.

A user wishing to exploit ZEPROM more fully by writing and loading M-code functions, or intending to operate ZEPROM in bank-switched mode, will require a more in-depth understanding of the HP-41 and M-code programming in particular. Although we have explained in some detail the subject of bank-switching within both user code and M-code programs, M-code programming requires a complete handbook to itself and is thus beyond the scope of this handbook. Users interested in M-code programming or synthetic programming should refer to the handbook for the ZENROM Module.

This ZEPROM Programmer's Manual covers the following topics:

- a brief overview of the format used for HP-41 ROM modules,
- the organisation of ZEPROM, its possible configurations and how to select the best for your circumstances,
- possible burning methods, fixtures and software,
- bank switching in both M-code and user code routines,
- burning using the ZEPROM Voltage Converter,
- the functions provided in the PROGRAMMER ROM software.
- using the automated utility routines; enabling inexperienced users to load programs from memory directly into ZEPROM.

Because programming skills and requirements vary so widely, it is difficult to detail a sequence for reading this manual; however, we strongly recommend that you do read all chapters. Even though you may not fully understand each topic, you will find the background information of benefit when you eventually come to burn ZEPROM with your own programs. In writing the manual, we have tried to provide program examples to illustrate the procedures involved. By examining those examples, users should very quickly be able to burn and use their own modules.

Throughout this manual we have referenced HP-41 memory address locations, etc., in hexadecimal notation.

Chapter 1

The HP-41 ROM Format

This chapter is intended only as a brief summary of the subject. For a more detailed explanation, users should refer to the ZENROM Owner's Manual, or one of the many text books on the HP-41.

The HP-41 can address 65536 (64K) words of ROM and each word in that ROM space has a 4 digit (16 bit) address at which it is located. This 64K of ROM space is split up into 16 *pages* of 4K-bytes each, of which pages 0 through 7 are reserved for the HP-41 itself or for system enhancements such as Timer, HP-IL module, etc..

The eight pages 8 through F are dedicated to the four I/O ports at the rear of the computer and each port contains two of the 4K-byte pages; thereby allowing either 4 or 8K ROM modules to be plugged into the port. However, most HP application modules are 4K and as such occupy only the lowest addressed page available in that port. A few ROMs, such as the HP-IL DEVELOPMENT ROM, contain 8K of data and so occupy both pages of that port.

Port 1	**Port 2**
Upper Page (9-hex)	*Upper Page (B-hex)*
Lower Page (8-hex)	*Lower Page (A-hex)*
Port 3	**Port 4**
Upper Page (D-hex)	*Upper Page (F-hex)*
Lower Page (C-hex)	*Lower Page (E-hex)*

HP-41 Input & Output Ports

This limited address space for each port places restrictions on the use of the high capacity modules such as ZEPROM, in that because the module contains 16K, it will electrically occupy *two adjacent* ports, even though it physically occupies only one port. For example, if plugged into port 1 *or* 2, it consumes both port 1 *and* 2 address space. Similarly, plugged into port 3 *or* 4, it uses *both* ports 3 *and* 4. It was in order to overcome these limitations that Hewlett-Packard developed the technique of 'bank-switching' in which only 8K of the module is switched on-line at any one time. These switched banks, called 'Primary' and 'Secondary', are shown in the following HP-41 Memory Map. Bank-switching will be described in more detail in Chapter 3.

Primary Bank		*Page*	Secondary Bank		*Page*
Port 4	*Upper Page*	**F**	**Port 4**	*Upper Page*	**F**
	Lower Page	**E**		*Lower Page*	**E**
Port 3	*Upper Page*	**D**	**Port 3**	*Upper Page*	**D**
	Lower Page	**C**		*Lower Page*	**C**
Port 2	*Upper Page*	**B**	**Port 2**	*Upper Page*	**B**
	Lower Page	**A**		*Lower Page*	**A**
Port 1	*Upper Page*	**9**	**Port 1**	*Upper Page*	**9**
	Lower Page	**8**		*Lower Page*	**8**
HP-IL/Mass Storage ROM		**7**			
Printer ROM		**6**			
Timer ROM		**5**	**CX Extended Functions**		**5**
Reserved page		**4**			
CX Extended Functions		**3**			
Operating System		**2**			
		1			
		0			

HP-41 Memory Map

If no module is present in any particular page, then that page will appear to the HP-41 to be full of NOPs (000 words).

All pages in ROM or ZEPROM must conform to a specified format irrespective of whether they contain M-code functions or user code programs. This is shown in the diagram below. In referring to addresses in the page, we will use a general format, e.g. *x*FC7h, where the '*x*' refers to the particular page number of the port into which the module is plugged, and the 'h' refers to the value being expressed in hexadecimal notation.

The first word of each 4K page, at address *x*000h (hex), is the XROM number of that page, coded in hexadecimal. For example the first word of the TIMER ROM, which has an XROM number of 26, will be 01Ah. The maximum value of this word is 01Fh, i.e. XROM 31. The second word of the page, at address *x*001h, indicates the number of catalogue entries in that ROM. (The header of a page, e.g: -TIME 2C, also counts as an entry in the catalogue.) Once again this word is coded in hexadecimal and can range in value from 000h, for no catalogue entries, to 040h for 64 catalogue entries.

Address	Description
x000	XROM number of page
x001	Number of FAT (catalogue) entries (n)
x002 to x003	Address of the first function in page
:	
:	
x(2n) to x(2n+1)	Address of the last function in page
x(2n+2) to x(2n+3)	End of the FAT (signified by two null words)
x(2n+4)	Start of functions and/or program code
:	
:	
xFC7 to xFCA	Bank-switching code (required in switching pages)
:	
:	
xFF4 to xFFA	Interrupt vectors
xFFB to xFFE	ROM trailer and bank-switched bits at *x*FFD
xFFF	Page checksum

HP-41 ROM Format

The next section of the page is called the 'Function Address Table', commonly known as the 'FAT'. Each entry in the catalogue requires two words in the FAT to determine the start address of the function (or program) and other status information about that entry. Because the HP-41 needs to know where each module function or program is located in order to be able to execute it, the start addresses are stored in the FAT. The FAT itself is arranged in catalogue order, with the first pair of words pointing to the address of the first catalogue function, the second pair pointing to the second function, etc.. The last entry in the FAT must be followed by two NOPs (000h) marking the end of the FAT. For example, a page with two functions might appear as follows:

Address	Word	
x000	015	XROM number of page is 21 decimal
x001	002	Page has only 2 catalogue entries
x002	004	
x003	02F	First function starts at address x42Fh
x004	001	
x005	023	Second function starts at address x123h
x006	000	
x007	000	NOPs to mark the end of the FAT

Normally the first function in a page will be the header, which is coded in the same manner as any other function. A header should always be at least 8 characters in length so that it cannot be executed by 'conventional' methods and so that it will show up as a ROM page header during a CAT 2 on an HP-41CX. It is customary for the first executable address of the header to be a RTN and for the header name to begin with a hyphen.

The rest of the page, up to the special reserved words starting at address *x*FF4h, is available for function or program code. However, if the page is to be used for bank-switching, then the special bank-switch code defined by Hewlett-Packard to swap the banks of a bank-switched module will be loaded at addresses *x*FC7h through *x*FCAh.

Addresses *x*FF4h through *x*FFAh are reserved for interrupt vectors which are polled by the operating system at various times. Unless you fully understand the use of these interrupts, we recommended leaving them as NOPs since their misuse can cause the HP-41 to lock up.

After the interrupt vectors come 4 words, at addresses *x*FFB through *x*FFE, containing the page trailer. The trailer should contain a four character sequence indicating the code revision number. In addition, the 8th and 9th bits of the word at *x*FFD will define whether or not that page is bank-switched.

The last word of a page, at address *x*FFF, is always a page checksum.

Chapter 2

The ZEPROM Module

Physical Description

ZEPROM is a 16K EPROM (Erasable Programmable Read Only Memory) module for the HP-41 housed in a standard-sized module casing which plugs into the computer.

A ZEPROM module can be programmed and erased without dismantling the module, thus reducing the risk of damage to the circuitry. Erasure is achieved via a window in the top half of the casing above the EPROM chip by exposing it to an ultra-violet (UV) light source. A custom chip built into the module, interfacing the EPROM to the HP-41, also contains a mini-programmer to allow programming by a low-powered, low-cost device using suitable software. After programming, ZEPROMs can only be erased by exposure to an intense UV light source. For specifications, etc., see Appendix A.

ZEPROM complies with Hewlett-Packard's convention for bank-switching and uses the machine code instructions ENBANK1 (100h) and ENBANK2 (180h) for control of switching. Because many other devices and software for M-code programming all use the WMLDL (040h) instruction for programming, we have designed ZEPROM to be programmed in the same way. However, because EPROM type devices require a longer write cycle, a special algorithm is needed to ensure that they stay programmed once the programming voltage is removed. The algorithm is detailed in Appendix A.

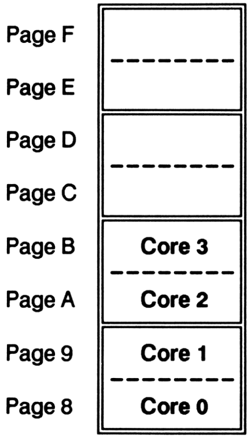
ZEPROM Configurations

ZEPROM can be configured and programmed to appear to the HP-41 in a number of different formats. The most suitable format will be determined by the user's own particular requirements and is set under software control to give the greatest flexibility. A ZEPROM module has two basic configurations, 'bank-switched' and 'straight' (unswitched); these being controlled by data burnt into ZEPROM at the location *x*FFDh in each module core. By using a bank-switched format and duplicating code into more than one core, ZEPROM can emulate various other configurations.

Straight 16K Module

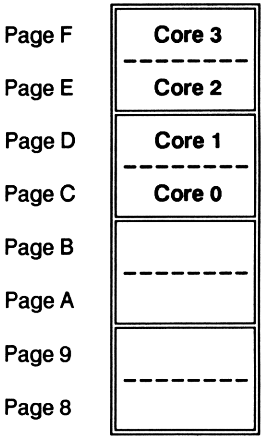
In its basic form, ZEPROM is configured as a 16K module, split into four 4K pages. This is the default mode in which ZEPROM is supplied and into which it is forced by the ZVC when programming. A 16K straight module occupies the address space of *two* adjacent ports, no matter which port it is plugged into.

If you consider ZEPROM as consisting of four separate memory cores of 4K each, the module would appear in the following memory locations:



Page F	
Page E	
Page D	
Page C	
Page B	Core 3
Page A	Core 2
Page 9	Core 1
Page 8	Core 0

ZEPROM in Ports 1 or 2



Page F	Core 3
Page E	Core 2
Page D	Core 1
Page C	Core 0
Page B	
Page A	
Page 9	
Page 8	

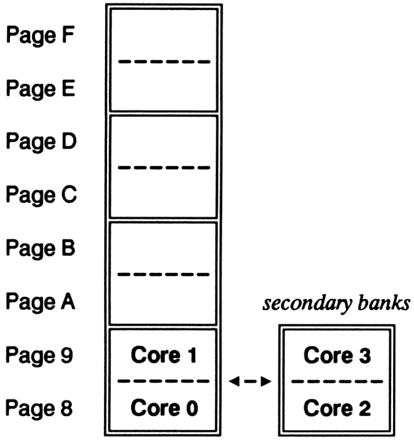
ZEPROM in Ports 3 or 4

Bank-Switched 16K Module

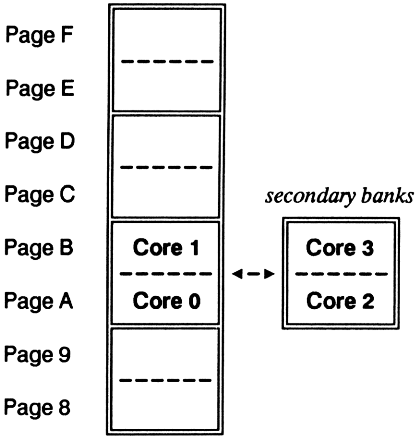
In a bank switched module, only two of the four 4K cores (the primary banks) are normally enabled (on-line) at any one time; the other two (secondary banks) being disabled (off-line). The enabled and disabled cores can be interchanged when needed under program control.

The advantage of this bank-switching facility to the user is that the module only consumes the address space of one port, rather than the two of a straight 16K module. In this way up to 4 ZEPROMs can be plugged into an HP-41 to give a total capability of 64K.

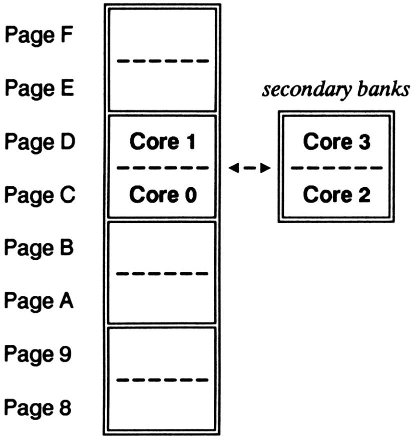
It is important to note that if you swap the banks of a module in any one port then that is the only module that is affected. Bank-switching instructions act only on the module in which they reside.



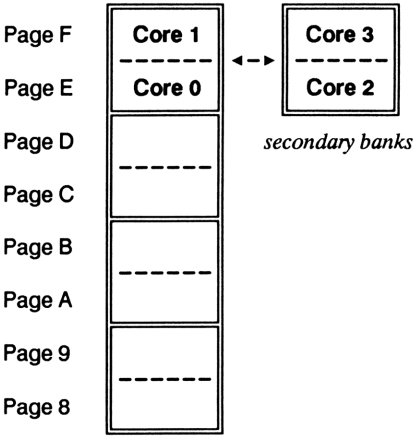
ZEPROM in Port 1



ZEPROM in Port 2



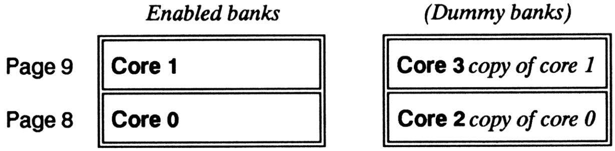
ZEPROM in Port 3



ZEPROM in Port 4

Straight 8K Module

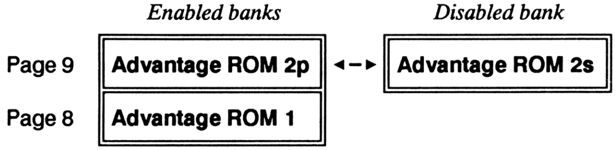
ZEPROM can also be programmed to emulate a normal straight 8K module, such as many of the application ROMs supplied by Hewlett-Packard. The straight 8K module is in fact a special example of a bank-switched 16K module in which, although configured as bank-switched, the programmed software never swaps the banks. However, we recommend that the secondary banks are programmed with the same code as their primaries, rather than being left blank - just in case a coding error ever causes the banks to switch. If you intend using this mode, you should bear in mind that the ZEPROM module is supplied by us with the PROGRAMMER ROM software already loaded into core 3.



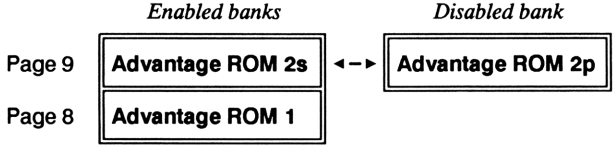
Bank-Switched 12K Module

In addition to the 16K bank-switched mode, ZEPROM is fully code-compatible with Hewlett-Packard's own 12K bank-switched ROM module. This means that programs developed using ZEPROM can easily be transferred to an HP custom ROM without modification.

To explain the 12K bank-switched ROM format as used by Hewlett-Packard, let's consider an HP Advantage ROM plugged into Port 1. The following diagram shows its configuration:

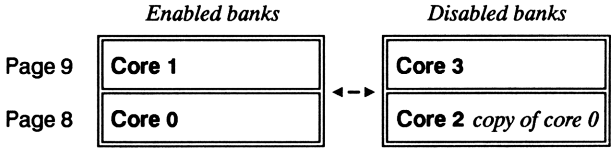


The lower half of the port is always occupied by ROM 1. However, the upper half can be occupied by either ROM 2p (primary) or ROM 2s (secondary) depending on which is required at that time. When power is first applied to the module, ROM 2p will reside in page 9 and ROM 2s is disabled. If you execute an Advantage ROM function which requires access to code in ROM 2s then that function will execute the machine code instruction to enable ROM 2s which will then appear in page 9 and ROM 2p will in turn be disabled:

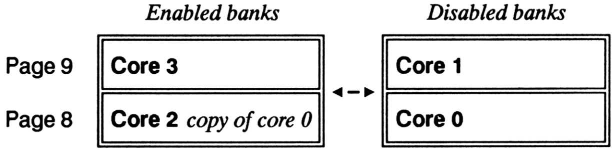


In showing an HP 12K ROM, you will notice we haven't shown a secondary bank for ROM 1 in the lower page (page 8). This is because an HP module doesn't contain memory at that location. Instead ROM 1 is non-switching and will *always* be available, no matter whether bank ROM 2p or ROM 2s is currently enabled.

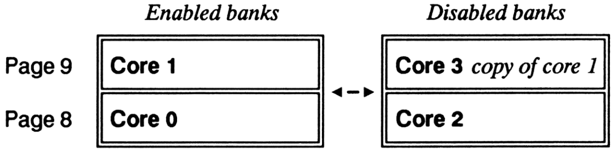
Because ZEPROM has four cores and both of its secondary banks always swap together, we use a slightly different technique to program ZEPROM in 12K bank-switched mode. This involves programming the secondary bank of the lower page (Core 2) with the same code as in the primary bank (Core 0). This means the HP-41 always has access to the code, no matter which bank is enabled.



When software switches the ZEPROM module by the ENBANK2 instruction, both secondary banks will be enabled:



The above example illustrates emulating a 12K ROM with the upper page being switched (an HP Advantage ROM). However, ZEPROM can also emulate a 12K bank-switched ROM with its lower page switching. In this case, core 3 would be programmed to match core 1:



Selecting the Best Configuration

The setting of a ZEPROM to straight or bank-switched configuration is controlled by the upper two bits (bits 9 and 8) of the word burnt into ZEPROM at address location *x*FFD in core 1 and core 3 of the module. When first plugged into the HP-41, these bits determine whether the module is seen as straight or bank-switched. If the bits are zero then the ZEPROM is configured as a straight 16K module; if they are non-zero then the ZEPROM is bank-switching. This is procedure is similar to that used in HP custom ROMs in that the upper two bits at address *x*FFD indicate if that page is bank-switched.

There are three main factors to consider when deciding how to best configure your ZEPROM:

1. The total size of all your programs or data that you want to burn into it,
2. The configuration of the rest of your HP-41 system, i.e. the other modules plugged in, and
3. How well you understand the techniques of bank-switching.

Software less than 8K

If the total size of all your program and data is less than 8K-bytes, then the simplest solution is to configure the ZEPROM as a straight 8K module.

An 8K module has the advantage that it will only occupy the address space of the port into which it is physically plugged and therefore does not conflict with other modules that you might have plugged into adjacent ports. It will also mean that you don't have to worry about incorporating any special bank-switching instructions and thus your programs will not need to be modified in order to run from ZEPROM.

Note, however that in order to emulate a straight 8K module, you actually configure ZEPROM as being bank-switched (with bit 8 of the data at addresses FFD in all pages being set). This forces the module to occupy only 8K of address space (one port). However, it is very advisable, though not obligatory, to also duplicate your entire 8K of programmed data into the module's secondary banks. In this way, if your software does ever cause the banks to swap by mistake, there will be no effect.

In deciding to use ZEPROM in 8K mode, you should bear in mind that you may have problems if you later need to add a program to the ZEPROM. If there is not enough room in the 8K, you may have to erase your ZEPROM, reconfigure it to a larger size, and then reburn all your programs. Also bear in mind that ZEPROM is supplied with the PROGRAMMER ROM software already burnt into core 3 and that this is lost in erasing.

Software between 8K and 12K

If you have, or envisage having, between 8K and 12K of programs and data to put into a ZEPROM then your options are limited to using either a 12K configuration, a 16K straight or a 16K bank-switched configuration. However, if you wish to keep the PROGRAMMER ROM in your ZEPROM module, then you cannot use the 12K option; since it is incompatible with a 12K configuration. If you are intending to use other burning methods and software, and so do not require the PROGRAMMER ROM, you can erase the entire module and thus use the 12K configuration.

Using a 12K format has a slight advantage over the 16K bank-switched mode in that it is slightly easier to develop bank-switching code. This is because there is a 'permanent' 4K block that is always on-line and as such you can always call, or return to that block without having to consider which bank is enabled. Another advantage over the straight 16K mode is that it only occupies the address space of the port into which the module is plugged and therefore does not affect the rest of your HP-41 system.

Software between 12K and 16K

If you do not mind consuming two adjacent ports with ZEPROM, then the straight 16K is the simplest format to use since there is no need to alter your software to incorporate bank-switching and all programs or functions are always readily available.

Although a straight 16K mode does consume two ports, if you have either a Memory module, Extended Memory Module, Timer ROM, Printer or HPIL module in the adjacent port, there will be no conflict. Such modules are classified as 'system' modules and therefore do not occupy a port address. The ZEPROM module comes supplied in this straight 16K mode and is also forced into it whilst programming is in progress by the ZEPROM Voltage Converter programming fixture.

Up to 16K with all ports needed

The 16K bank-switched configuration is ideal if you have a lot of data to load into ZEPROM, yet only want to use up the address space of one port. It is, however, by far the most difficult to program for, as you must very carefully organise and plan your code in advance. This is necessary to ensure that the correct module bank is always enabled, and that the code in the other bank begins at the specific address where you switched the banks. In addition to this, you will have to modify your existing programs and functions to include the bank-switching instructions.

Before deciding to use a bank-switched configuration, make sure that you have a full understanding of bank-switching and its consequences.

Chapter 3

Bank Switching

Introduction

Most computers have an address space in which they store their data and programs; this space having a limit on the total number of bytes it may contain. One common method to overcome this constraint is to arrange for multiple banks of memory to appear in the same address locations, and then have some hardware arbitration scheme which makes only a single bank available to the computer at any one time. This process, known as 'bank switching', allows a significant increase in memory capacity with only a small increase in program complexity.

Benefits:

The principal benefit of bank-switching is the expansion of memory it offers; instead of being limited to 32K of available port address space, by bank-switching all plug-in modules, 64K becomes available. An additional benefit for the commercial software developer is that of making programs and data stored in modules harder to 'crack'.

Disadvantages:

Because the HP-41's bank-switching scheme is an add-on to the memory system, programming support for it in the operating system is nonexistent; you have to do everything yourself. Even for the programmer working at machine code level, this can lead to some fairly complex code if you don't plan things carefully in advance. For the user code programmer, bank switching has only now become possible with the advent of ZEPROM and the PROGRAMMER ROM. However, although the PROGRAMMER ROM functions make user code bank switching possible, its use does require careful planning and a good understanding of the HP-41 and its ROM structure.

Implementation on the HP-41

On the HP-41, bank-switching was an after-thought to the original system design. As such, it does not integrate as smoothly into the system as one would perhaps like; for example, the operating system offers no support for bank-switching memory. Nevertheless, where an application requires it, it does offer the ability to increase the addressable memory by doubling the total amount of memory capable of being connected to the machine (via its I/O ports), from 32K to 64K.

The HP-41 has four I/O ports at its rear; each providing the addressing connections for a plug-in module containing two pages of 4K-bytes each. It is normal to refer to these as being the upper and lower pages, or by their positions in the HP-41 memory map (pages 8h to Fh). See Chapter 1, or the *ZENROM Owner's Manual.*

Port 1 *Upper Page (9-hex)* *Lower Page (8-hex)*	**Port 2** *Upper Page (B-hex)* *Lower Page (A-hex)*
Port 3 *Upper Page (D-hex)* *Lower Page (C-hex)*	**Port 4** *Upper Page (F-hex)* *Lower Page (E-hex)*

In a bank-switched module, either one or both of the two 4K pages visible to the HP-41 in each port can be made bank-switching. For a bank switched ROM, there will be both a visible and a hidden bank for each single 4K page that allows switching. To make the banks swap over, Hewlett-Packard defined two spare CPU instructions as 'ENBANK1' and 'ENBANK2' to enable bank 1 and bank 2 respectively.

Bank 1 in a bank-switching page is often referred to as the primary bank, and bank 2 the secondary (or alternate) bank. Whenever a bank switching module is first plugged into the HP-41, its primary banks will always be visible. When an ENBANK2 instruction is issued by the module (not the HP-41 itself, which regards these instructions as harmless no-ops), it hides its primary bank and enables the secondary banks. The module remains this way until an ENBANK1 instruction is issued, or the module is removed from the HP-41 and then replaced.

In order to prevent all bank-switching modules from switching, the ENBANK1 and ENBANK2 instructions have been implemented such that they are only acted upon by the particular module that contains them. Whilst this causes some difficulties for the programmer, it does mean that all other modules ignore the instructions.

When writing bank-switching code for a particular module at an M-code level, you just put the ENBANK2 instruction in the code of that module, and when it executes, the module's secondary banks replace its primary banks (both 4K pages can swap). To allow one module to 'poke' another module into showing its alternate banks, HP have established a convention in which bank-switching instructions are placed at fixed locations within each 4K bank-switching page. By simply jumping direct to these locations, an M-code program can force another module to select a particular bank.

Bank Switching of Machine Code

We recommend two methods of implementing bank switching of ZEPROMs that contain mostly M-code functions and routines. One is ideal for situations where you have a few very large M-code functions; the other is ideal for the case where you have a very large number of smaller functions. As the latter is simpler to implement, we'll explain that first.

A Large Number of Small Functions

Consider the case where you have in excess of 200 short functions to put into a ZEPROM. Supposing the four pages of a ZEPROM plugged into port 1 of the 41 were initialised using the PROGRAMMER ROM functions [INITPG] and [ADDMCF], to contain 64 functions per page. [ADDMCF] adds the M-coded instructions to an already initialised page. ZEPROM will subsequently look like this:

	Primary Banks		**Secondary Banks**	
PAGE 8	**XROM 01**	*64 functions in page*	**XROM 03**	*64 functions in page*
	-ROM-1	*Page header*	-ROM-3	*Page header*
	PB01	} *These functions*	PB03	} *These functions*
	SB01	} *added by ADDMCF*	SB03	} *added by ADDMCF*
	:		:	
	:		:	
	:		:	
	:		:	
	bank switching	} *added by*	bank switching	} *added by*
	code	} *INITPG*	code	} *INITPG*
	RM1A	*Page trailer*	RM3A	*Page trailer*
PAGE 9	**XROM 02**	*64 functions in page*	**XROM 04**	*64 functions in page*
	-ROM-2	*Page header*	-ROM-4	*Page header*
	PB02	} *These functions*	PB04	} *These functions*
	SB02	} *added by ADDMCF*	SB04	} *added by ADDMCF*
	:		:	
	:		:	
	:		:	
	:		:	
	bank switching	} *added by*	bank switching	} *added by*
	code	} *INITPG*	code	} *INITPG*
	RM2A	*Page trailer*	RM4A	*Page trailer*

You can then fill up each of the four pages with function entries and code; trying as far as possible to keep the function code in the same page (and bank) as the function entry itself. When you wish to execute a function in your ZEPROM, you would perform the following steps from the keyboard or in a user code program:

- Execute the primary or secondary bank enable instruction in the currently enabled page, e.g. [PB01] or [SB01]
- Execute the desired function by name

Example:

Suppose your function called SOLVE resides in 'ROM-3' (page 8, alternate bank), and the primary bank is currently enabled. The steps to execute it are as follows:

Key Sequences	Description
[XEQ] [ALPHA] [S] [B] [0] [1] [ALPHA]	Execute [SB01] instruction to select the secondary bank of ZEPROM. [SB01] is used because, ROM-1 is currently enabled. The ZEPROM will switch banks, and enable the alternate bank.
[XEQ] [ALPHA] [S] [O] [L] [V] [E] [ALPHA]	Execute the M-coded function that you wish to use from that bank. After executing, it will exit with the alternate bank still enabled. To return to the primary bank, use ROM3's enable primary bank function ([PB03]).

In the case where you have only a few, very long functions to put into a ZEPROM, the recommended practice is slightly different.

A Small Number of Large Functions

Suppose that you have four M-coded functions, each of about 4K in length, to put into a 16K-ZEPROM module plugged into I/O port 1 of the HP-41.

Initialise both the primary banks in both pages (with [INITPG]) as being bank-switched, with three entries in each FAT (for two functions and the header). Instead of initialising the two secondary banks with [INITPG], just use [ADDBSW] to add the necessary bank switching code into both secondary banks and to set the bank-switched bit at FFDh. As can be seen from the diagrams, only the two primary banks now contain XROM numbers, page headers, FATs, etc.

Next add all the necessary bank-switching bytes (required by all bank-switching ROM banks) to both primary *and* secondary banks. The module will now look like the following:

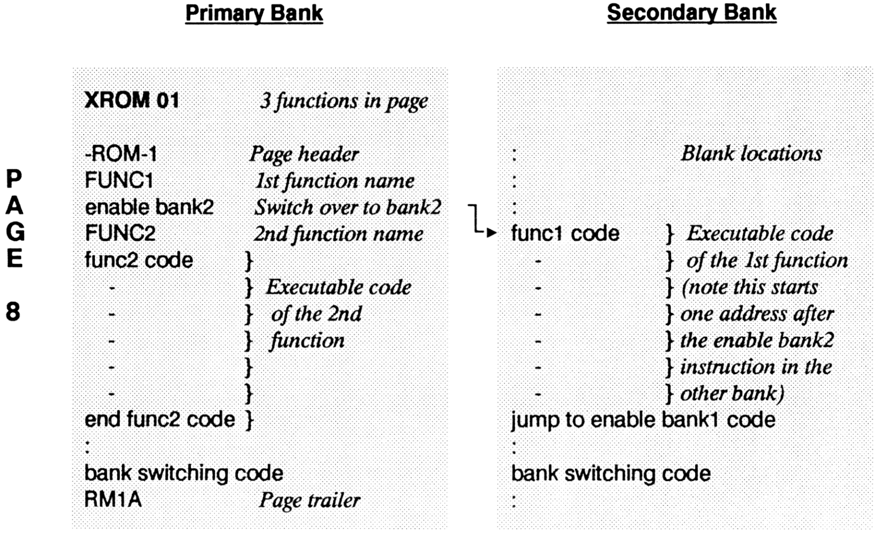
	Primary Banks	Secondary Banks
PAGE 8	**XROM 01** *3 functions in page* -ROM-1 *Page header* : : : bank switching code RM1A *Page trailer*	: : : : bank switching code :
PAGE 9	**XROM 02** *3 functions in page* -ROM-2 *Page header* : : : bank switching code RM2A *Page trailer*	: : : : bank switching code :

Instead of trying to spread the code for two functions over each bank, make the execution code for one of the functions (preferably the first) in the first bank just an enable secondary bank instruction. You can then follow this with the second function's code in its entirety. The real execution code of the first function is placed in the secondary bank. However, it must always start at one address after that at which the enable secondary bank instruction is located in the primary bank.

The reason for needing to align the code at corresponding locations in both banks is because of the way bank-switching is achieved on the HP-41. With more sophisticated computers, the computer can both bank-switch *and* begin execution from a different point in the new bank. With the HP-41, however, it just bank-switches and then continues execution from the address after the one it had reached in the previous bank. This is a restriction that applies whenever you switch banks in a running M-code function or user code program. To overcome this you need to plan your code carefully.

In writing your function for the secondary bank, you should end it in a different way. Normally, you would just use the M-code RTN instruction. If you did this with a bank-switching function, execution would halt with the secondary bank still enabled. Instead, you should replace the RTN instruction with a jump to the ENBANK1 code in that page - normally at location *x*FC7h (where *x* is the particular page number). This procedure will ensure that the primary bank is re-enabled when execution of the page-switched function is over.

This procedure is perhaps better explained by the following diagrams. These show only part of the module - the two banks of the lower page of port 1:



```
              Primary Bank                                   Secondary Bank

     XROM 01          3 functions in page

     -ROM-1           Page header                 :                   Blank locations
P    FUNC1            1st function name           :
A    enable bank2     Switch over to bank2   ┐    :
G    FUNC2            2nd function name      └─▶ func1 code      } Executable code
E    func2 code       }                              -           } of the 1st function
        -             } Executable code              -           } (note this starts
8       -             }  of the 2nd                  -           } one address after
        -             }  function                    -           } the enable bank2
        -             }                              -           } instruction in the
        -             }                              -           } other bank)
     end func2 code   }                           jump to enable bank1 code
     :                                            :
     bank switching code                          bank switching code
     RM1A             Page trailer                :
```

As this shows, this practice does leave unused areas in the ROM's bank, but you can always utilise them by having your code jump to those locations.

Let's now more closely examine the procedures involved in bank-switching of M-code by means of practical examples.

Examples

As explained earlier, it is beneficial to be able to execute all the ROM's functions without having to first enable the secondary bank. This entails grouping all FAT entries in the primary bank and putting the function's real executable code in the secondary bank. To do this, each secondary bank routines needs a *dummy* catalogue entry in the primary bank. Each of those entries points to a routine containing the following code:

```
         JNC     +01          Used only for compatibility with HP's ROMs
xAdr     ENBANK2              Enable secondary bank. Execution continues
                              from same location plus one in new bank.
```

Note: The `JNC +01` instruction is only necessary if you are writing code that may eventually be burned by Hewlett-Packard into a custom module. This because of a hardware error in the chip used by HP. If you are writing code specifically for ZEPROM, the instruction is not necessary, but for compatibility we recommend including the `JNC +01`.

No other code for the functions is needed in the primary bank; since execution will be in the secondary (alternate) bank.

The address of the line xAdr in the primary bank corresponds to one less than the start of the execution code in the secondary bank. This is because the banks switch between reading the ENBANK2 instruction in the primary ROM, and the reading of the next instruction from the secondary ROM. The execution process runs as follows, for simplicity, we have assumed that the primary bank is initially enabled (the default state when the ROM is first plugged in):

```
Addr   Primary code                Secondary code
       ...                         ...
8100   JNC    +01                  ???
8101   ENBANK2                     ???
       [banks switch at this point]
8102   ???                         1st byte of target routine
       ...                         ...
8110   ???                         last byte of target code
8111   ???                         *
8112   ???                         *
8113   ???                         NCGO   8FC7        3-word jump to enable
                                                      the other bank
       ...                         ...
8FC7   ENBANK1                     ENBANK1             enable other bank
       [banks switch at this point]
8FC8   RTN                         RTN
```

Note: In the above example, it is important note that the instruction at address 8113h (NCGO) is a 3-word jump. If a 2-word jump was used instead, the module containing this code would be port-dependent and would only work if plugged into a particular port.

In this example, '???' means "don't care", and the addresses are only examples. In practice, the code is arranged such that all the ENBANK2's are in a block, aligned with a block of local jumps on the secondary ROM side to the target routines, which may be just JNCs, or which may be three-word jumps (calls to GOL0..GOL3 in the HP-41 operating system, followed by a single-byte offset). Note that neither is a two-word jump, so it is not possible to interleave the JNCs and three-word jumps in one bank with the JNC +01 & ENBANK2 combination in the other without requiring padding bytes somewhere.

It is also usual to arrange ENBANK1 as a common exit routine to save having more than one ENBANK1 in the secondary ROM. This minimises the total number of ENBANK1 & ENBANK2 instructions that are required between the two ROMs, which, in turn, minimises the effort needed to align code in the two switching banks.

An Example:

Suppose our ROM has three routines that need to be bank-switched, because there is no room for them in the primary 4K bank. For this example, the primary bank has been initialised with the catalogue (FAT) containing just ten entries including those entries for our three routines called 'JOHN', 'BILL' and 'DAVE'.

Primary Bank Code: The following code should be loaded into the primary bank of the page.

```
toJohn 00B JNC    +01d
```

```
  8019 180 ENBANK2
toBill 00B JNC    +01d          Aligns with the line labelled 'fmJohn'
  801B 180 ENBANK2
toDave 00B JNC    +01d          Aligns with the line labelled 'fmBill'
  801D 180 ENBANK2

  801E 00E N                    Aligns with the line labelled 'fmDave'
  801F 008 H
  8020 00F O
  8021 00A J
  8022 3B3 JNC    toJohn        Catalogue entry for JOHN points here

  8023 08C L
  8024 00C L
  8025 009 I
  8026 002 B
  8027 39B JNC    toBill        Catalogue entry for BILL points here

  8028 085 E
  8029 016 V
  802A 001 A
  802B 004 D
  802C 383 JNC    toDave        Catalogue entry for DAVE points here
```

That's all that's needed in the primary bank.

Secondary Bank Code: In the secondary bank, the code which 'accepts' control when it is passed in from the primary bank should look as follows:

```
fmJohn 073 JNC     John         Loaded at 801Ah
  801B 000 NOP                  Just padding.  Never gets executed
fmBill 0B3 JNC     Bill
  801D 000 NOP                  Just padding.  Never gets executed
```

No need to jump to routine 'Dave', because we can start it here

```
fmDave 2E0 DISOFF               Switch display off whilst updating
  801F 3C1 *
  8020 0B0 NCXQ   2CF0          Clear & enable display
  8021 3BD *
  8022 01C NCXQ   07EF          Send message to display
  8023 004 D
  8024 001 A
  8025 016 V
  8026 205 E
  8027 0A3 JNC    Out           Jump to common exit routine

John   2E0 DISOFF               Switch display off whilst updating
  8029 3C1 *
  802A 0B0 NCXQ   2CF0          Clear & enable display
  802B 3BD *
  802C 01C NCXQ   07EF          Send message to display
  802D 00A N
  802E 00F H
  802F 008 O
  8030 20E J
  8031 053 JNC    Out            Jump to common exit routine

Bill   2E0 DISOFF                Switch display off whilst updating
  8033 3C1 *
  8034 0B0 NCXQ   2CF0           Clear & enable display
  8035 3BD *
  8036 01C NCXQ   07EF           Send message to display
  8037 002 B
  8038 009 I
  8039 00C L
  803A 20C L

Out    3BD *                     Start of common exit code
  803C 01C NCXQ   07EF           Send message to display
  803D 020
  803E 003 C
  803F 001 A
  8040 00C L
  8041 00C L
  8042 005 E
  8043 004 D
  8044 220                       Needn't left-justify with this space here
  8045 320 DISTOG                Restore/turn on the display
  8046 149 *
  8047 024 NCXQ   0952           Disable display
  8048 1F9 *
  8049 00C NCXQ   037E           Set the message flag
  804A 3AD *                     Re-enable primary banks, and exit
  804B 08C *
  804C 3C7 NCGO   8FC7
```

The address in both files can be practically anything you like, but the simplest place to put these bank-switching instructions is immediately following the catalogue, which means you need to work out the catalogue size for the ROM to figure out where to load the routines. In this example, we've assumed that the catalogue has ten entries (including these three functions), so the catalogue size is 2+10*2+2 [XROM+FCNS*entries+nulls], and so ends at 8017 (hex), so we have shown the example code loaded at 8018h in the primary bank. The code in the secondary bank now has to be positioned alongside the code in the primary bank, according to the alignment requirement specified in the code; hence, it would be loaded at 801Ah.

This would then give code arranged as follows:

```
Addr   Primary code                              Secondary code
                          program flow
8018   JNC    +01                                ???
8019   ENBANK2            ►-----┐                ???
801A   JNC    +01               └------►         JNC    John
801B   ENBANK2            ►-----┐                NOP
801C   JNC    +01               └------►         JNC    Bill
801D   ENBANK2            ►-----┐                NOP
801E   ???                      └------►         Dave code begins here
```

This procedure also copes with returning from the routine back to the primary bank by means of a 3-word jump (NCGO 8FC7) which will always be ENBANK1 & RTN instructions in all bank switched pages.

Following the flow of control while executing 'JOHN', the jump at 8018h jumps to the ENBANK2 at 8019h, which causes a bank-switch to occur before the instruction at 801Ah executes. This causes the JNC John to become visible as the next instruction, and execution continues with the remainder of the 'JOHN' code until the jump to 8FC7h causes the ENBANK1 instruction to switch banks back again to the primary set. This makes the RTN visible as the next (and final) instruction executed by the HP-41 as part of the 'JOHN' code, and leaves the primary banks enabled. A similar thing will happen when 'BILL' is executed. 'DAVE' only differs in that there is no jump on the secondary side and execution continues directly with the DISOFF instruction.

For many simple applications, this scheme will suffice, although its use does require careful planning.

If your function should need to force a different module to switch its banks, you can use the following subroutines. These select the primary [or secondary] bank of a given ROM page. They do this by exploiting a feature of the Hewlett-Packard bank-switching specification; which is that the four addresses from *x*FC7 through *x*FCAh in any bank-switching page must contain the four instruction sequence:

```
:
xFC7  ENBANK1
xFC8  RTN
xFC9  ENBANK2
xFCA  RTN
:
```

This code is loaded at the same location in both the primary and secondary banks which switch with one another. Being therefore always visible to the HP-41, it can be used to force selection of either the primary or secondary banks at any time. The following two subroutines, which expect to be called from another piece of code, will force selection of the page specified by digit '6' of CPU register 'C'. In M-code programming, this is normally represented as 'C[6]'.

Note: these routines rely on the pages being bank-switched and having the bank-switch code at locations *x*FC7 through *x*FCA. In order to check if a page is bank-switching, you should check that one of the top two bits of the word at location FFD in that page is set. In these example routines, we have assumed the ROM is bank-switching, and therefore haven't checked it.

BANK1N and BANK2N Routines

Input:	C[6] -- page number for ROM to select.
Output:	Desired ROM with bank1 [bank2] enabled, C[6:3] is the execute address of routine called in selected ROM to perform bank-switch.
Uses:	C[5:3]
Assumes:	ROM page specified actually exists, *and* is bank-switching

WARNING: **Do not** GOTO (branch) to these routines - only call them!

Purpose: Bank1N forces the specified bank-switching ROM to have its primary banks enabled.

Bank2N forces the secondary bank to be enabled.

The call to the alternate bank is done by jumping to address *x*FC7h in that ROM to enable its primary banks, and *x*FC9h for the secondary banks when returning to the caller for exit.

If you want to incorporate this code into some other routine, and can't cope with the operating system overhead of the subroutine call, you will need to make you own arrangements for pushing the appropriate return address onto the stack.

BANK1N Code:

```
Bank1N 03C RCR    3             C[3] = = ROM page number
       130 LDI
       038 CON    038           inverse of FC7
       033 JNC    +04           jump to common code
```

BANK2N Code:

```
Bank2N 03C RCR    3             C[3] = = page number
       130 LDI
       036 CON    036           inverse of FC9
```

At this point, C[3] contains the page address of the ROM which we want to use, and C[X] the bit-wise inverse of the jump address.

```
       2A6 C=-C-1  X            find true address in that page
       1BC RCR     11           rotate final address into C[6:3]
       1E0 GTOC                 do the call
```

At this point the ENBANK1 [ENBANK2] instructions will have been processed, so the primary [secondary] ROM bank will have reappeared, and the secondary [primary] ROM bank will have disappeared. The ENBANK1 [ENBANK2] is followed at *x*FC8h [*x*FCAh] by a RTN, which pops the return address of the function which called this code.

Bank Switching of User Code Programs

Theoretically, bank-switching of user code programs is considerably more complicated than performing bank-switching in M-code. The reason being that the means of swapping banks is really only available at a machine code level, and that there are no user code equivalents to the ENBANK1 and ENBANK2 M-code instructions.

However, in designing ZEPROM and its associated products, the design team decided that the ability to switch banks should also be available within user code programs. The PROGRAMMER ROM software, provided pre-programmed in one bank of ZEPROM, therefore supplies functions that allow the user to accomplish this.

Because of the nature of bank-switching, as discussed earlier in this chapter, bank-switching instructions can only operate on the particular module in which they reside. For this reason, each page, and the switched banks within that page, must contain the functions necessary to perform bank-switching. These functions are loaded into an initialised page by executing [ADDUCF] and [ADDMCF] from the PROGRAMMER ROM.

[ADDUCF] loads three special functions [BGTOxx], [BXEQxx] and [BRTNxx]; which are the bank-switching equivalent of the normal GTO, RTN and XEQ. [ADDMCF] loads the functions [PBxx] and [SBxx] which allows enabling of the primary or secondary banks as desired. In each case, the 'xx' in the function's name represents the XROM-ID of the page into which the function is loaded. These unique numbers are necessary as an identification of the particular page or bank in which the functions reside and thus provide a means of specifying which ZEPROM is to switch its banks. Although these XROM-identities are used for all HP-41 plug-ins, they are normally seen only when you assign a module function to a key, or enter it in a program, then remove the module.

Whenever a user code program is loaded into a ZEPROM or ROM, all global alpha labels in that program are given an entry in the Function Address Table (FAT) for that ROM. This means that all such labels will also take on an XROM number; this being a combination of the ROM's XROM-ID and the FAT entry number within that ROM. Normally, if your program is to branch to an alpha label, you simply key in the instruction, e.g. `GTO 'ABC`. When the program is run the 41 will search through CAT 1 followed by CAT 2 until it finds the label corresponding to the GTO. An XEQ is slightly different in that, when the program is typed in, the function is loaded as either `XROM 'ABC`, or `XEQ 'ABC` depending upon whether or not the label was found in ROM. When run, the program either searches for the XROM number or searches through CAT 1 and CAT 2, label depending upon how the function was stored.

If the target label of a GTO or XEQ is in a bank-switched ROM and that bank is not selected (enabled) at the time of execution, then the label will not be found since the built-in GTO and XEQ functions cannot bank-switch. It is for this reason that the PROGRAMMER ROM provides [BGTOxx] and [BXEQxx] functions. However, since both functions expect an alpha label in Reg X, a restriction of six characters is placed on the length of any such label. In operation, [BGTOxx] and [BXEQxx] switch banks and then execute either GTO IND X or XEQ IND X for that label. It is perhaps worth pointing out that these functions could not be made prompting, as per the ordinary GTO and XEQ, since the 41 operating system dictates that external functions which prompt for an argument must be non-programmable.

Because BXEQxx and BGTOxx cause the banks to swap before calling the label, a special RTN function is also required that first swaps the banks and then performs the RTN. This function is BRTNxx.

Examples

Consider the following situation where a ZEPROM module is configured as 16K bank-switched. Each page has previously been initialised with INITPG and ADDUCF

	Primary Banks		Secondary Banks	
	XROM 01		**XROM 03**	
	-ROM-1	*Page header*	-ROM-3	*Page header*
P	BGTO01	} *These functions*	BGTO03	} *These functions*
A	BRTN01	} *added by ADDUCF*	BRTN03	} *added by ADDUCF*
G	BXEQ01	}	BXEQ03	}
E	:		:	
	:		:	
8	:		:	
	bank switching	} *added by*	bank switching	} *added by*
	code	} *INITPG*	code	} *INITPG*
	RM1A	*Page trailer*	RM3A	*Page trailer*
	XROM 02		**XROM 04**	
	-ROM-2	*Page header*	-ROM-4	*Page header*
P	BGTO02	} *These functions*	BGTO04	} *These functions*
A	BRTN02	} *added by ADDUCF*	BRTN04	} *added by ADDUCF*
G	BXEQ02	}	BXEQ04	}
E	:		:	
	:		:	
9	:		:	
	bank switching	} *added by*	bank switching	} *added by*
	code	} *INITPG*	code	} *INITPG*
	RM2A	*Page trailer*	RM4A	*Page trailer*

Now suppose we load one user code program into each core; `'PROG1` into page 8 primary, `'PROG2` into page 9 primary, `'PROG3` into page 8 secondary and `'PROG4` into page 9 secondary. The ZEPROM module will now appear as in the following diagram:

	Primary Banks	Secondary Banks
PAGE 8	**XROM 01** -ROM-1 BGTO01 BRTN01 BXEQ01 'PROG1 } *user code program* : : bank switching code RM1A	**XROM 03** -ROM-3 BGTO03 BRTN03 BXEQ03 'PROG3 } *user code program* : : bank switching code RM3A
PAGE 9	**XROM 02** -ROM-2 BGTO02 BRTN02 BXEQ02 'PROG2 } *user code program* : : bank switching code RM2A	**XROM 04** -ROM-4 BGTO04 BRTN04 BXEQ04 'PROG4 } *user code program* : : bank switching code RM4A

Let's suppose that we have written our programs in the following manner with 'PROG1 calling both 'PROG2 and 'PROG3 as subroutines. The code for 'PROG1 will thus have the following structure:

```
LBL "PROG1"
:
XEQ "PROG2"               Call 'PROG2 as normal sub-routine
:
"PROG3"                   Load label of bank-switched routine
ASTO X                    Store it into Reg X
BXEQ01                    Call 'PROG3 as bank-switched
:
END
```

Notice that, in this instance, there is no need to call 'PROG2 as bank-switching, since we know that 'PROG2 must be in the currently selected bank because the calling program is also in that bank. However, in order to call 'PROG3, its label must be loaded into Reg X and the bank-switching XEQ function (**BXEQ01**) used since the program resides in a different bank. It is particularly important to note that the function used is BXEQ01 and not BXEQ03. This is because you must always use a bank-switching function resident in the currently enabled bank - the target bank's functions being effectively invisible until that bank is enabled.

The code for 'PROG3, loaded in a secondary bank, would be written as follows:

```
LBL "PROG3"          Notice the use of BRTN03 and not
:                    BRTN01 to bank-switch RTN to the
:                    primary bank since secondary bank
BRTN03               will be enabled at that time
```

Now let's suppose that 'PROG4 also needs to call 'PROG3. If 'PROG3 is left as above (with a BRTN03 instruction at its tail), then a call by 'PROG4 will return execution to the wrong place since it will have swapped the banks over.

It is therefore necessary to establish a protocol by which a subroutine can determine if it was called from the same or another bank. The simplest way to do this, is by means of a flag:

```
LBL "PROG1"
:
XEQ "PROG2"          Call 'PROG2 as normal sub-routine
:
"PROG3"              Load LBL of switched routine
ASTO X               Store it into Reg X
SF 00                Set flag to indicate bank-switch call
BXEQ01               Call 'PROG3 as bank-switched
:
END
```

```
LBL "PROG4"
:
CF 00                Clear flag to indicate non-switched
XEQ "PROG4"          Call 'PROG4 as an ordinary XEQ
:
END
```

```
LBL "PROG3"
:
:                    Test flag to determine whether
FS? 00               called from another bank?
BRTN03               Yes, then do a bank-switched RTN
RTN                  No, so do an ordinary RTN
```

In the preceding example, we have only used ADDUCF to add the user-code functions BGTOxx, BRTNxx and BXEQxx in the switching banks. In cases where you intend loading all your sub-routines into the secondary bank and will only call them via the main programs in the primary bank, this will suffice. However, it is also possible that you may, for space or other reasons, need to insert some main programs into the secondary bank. *See WARNING below.* Executing main programs in the secondary bank can be achieved by either of the following means:

1. Enabling the page's secondary bank with SBxx, then
   Executing the desired program by name with XEQ, and finally
   Re-enabling the primary bank with PBxx. (SBxx and PBxx are added to an initialised page by ADDMCF.)
2. Executing the desired program by name with BXEQxx.

Suppose a program, called ᵀPROG3, resides in ROM-3 (page 8, secondary bank) and the primary bank is currently enabled. You could execute it from the keyboard as follows:

Key Sequences	Description
XEQ ALPHA S B 0 1 ALPHA	Enables the secondary bank. Note: SB01 is used because, ROM-1 is currently enabled.
XEQ ALPHA P R O G 3 ALPHA	Executes program in that bank and exits with the secondary bank still enabled.
XEQ ALPHA P B 0 3 ALPHA	Re-enable the primary bank. Note: PB03 is used because, ROM-3 is currently enabled.

!
Warning

Exercise **extreme care** when executing programs in other banks either from the keyboard or from another program. Before doing so, you must be absolutely sure how that program is structured. SBxx and PBxx should also **only** be used in a running program when the code in both primary and secondary banks of that page is exactly identical (e.g. in emulating a 12K module) since otherwise the banks switch and the next program instruction is no longer available.

When either of the two methods discussed are executed from the keyboard, the called program will either terminate with the secondary bank enabled (if normal RTN/ENDs were used) or swap back to the primary bank (if BRTNxx was used).

If BXEQxx is executed within a running program, then the called program will either swap banks and correctly continue execution (if BRTNxx was used), or will stay in that bank, but return to an address within that bank that has totally unrelated code in it (if a normal RTN/END was used). The result of this is absolutely unpredictable.

The need to use correct terminating returns is dictated by the HP-41 method of bank-switching. More sophisticated computers can both bank-switch *and* recommence execution from a different address, however, the HP-41 just bank-switches and blindly continues execution from one address after that it had reached in the previous bank. **To overcome this you need to plan your bank-switched programs very carefully.**

Chapter 4

ZEPROM Voltage Converter

The ZEPROM Voltage Converter, or ZVC, has been designed as a companion product to ZEPROM and provides the user with an inexpensive, simple-to-use means of burning (programming) ZEPROMs when used with software such as the PROGRAMMER ROM.

The ZVC is a single unit, bare board device protruding from the back of a standard HP-41 module housing. This housing allows the ZVC to be plugged directly into one of the four I/O ports at the back of the HP-41. The contacts at the other end of the ZVC are provided for plugging the ZEPROM module onto.

When plugged into the HP-41, the ZVC increases the 6V supply available from the HP-41's batteries to the 12.5V needed to burn ZEPROM.

Switching ON & OFF

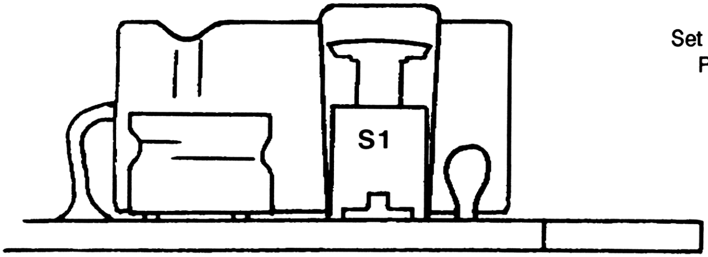
The ZVC contains a small vertically moving switch (S1) next to the connector for attaching a ZEPROM module. This switch, used in programming, has two purposes:

To activate the 12.5V programming voltage

To force the attached ZEPROM module into a *straight 16K* mode. This applies even if that particular ZEPROM module has already been part-burnt and configured as bank-switched, etc.. Forcing the ZEPROM into 16K mode is necessary in order that the HP-41 can see all four 4K-cores in the module during programming.

Programming Switch

Set to OFF Position



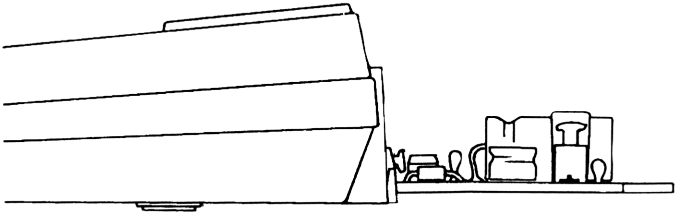
The programming switch is activated (turned ON), by pushing the switch actuator downwards. To turn the switch OFF, lift the actuator upwards with a fingernail. The direction of movement to turn the switch ON is marked on one side of the switch body.

Plugging in

The ZVC plugs into any of the HP-41's I/O ports in the same manner as a normal module. The correct orientation, with the HP-41 resting on a table, is with the ZVC's components facing uppermost - *see diagram below*. In this orientation, the ZVC can be gently slid into the port.

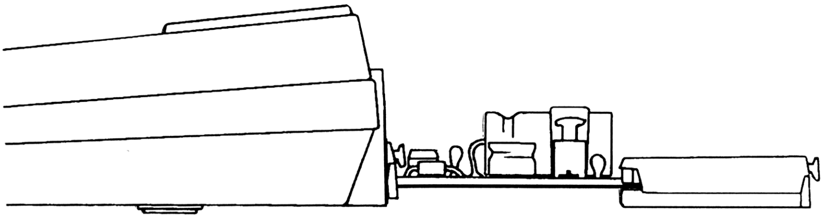
!
Cautions

- The HP-41 computer must be switched OFF before modules are plugged into or removed from it. This also applies whenever the ZVC device is inserted or removed, and when a ZEPROM module is attached to or removed from the ZVC. Failure to do this may cause the computer to reset, causing a loss or corruption of data in memory.
- To insert or remove a ZVC, hold it by its edges between thumb and forefinger. Avoid holding the device by its contacts or underneath.
- The ZVC can only be inserted one way into an HP-41 port. If any obstruction is felt, the orientation is most likely incorrect. Never force the ZVC into a port as this can damage contacts in either or both devices.
- Protect the HP-41's ports from dust by keeping a port cap installed in any empty port. Do not place fingers, tools or other foreign objects into the ports. Such action can cause serious damage to port contacts and/or internal circuitry.



With the ZVC plugged into the HP-41, the ZEPROM module can be attached to the ZVC. Before attaching a ZEPROM, ensure that the ZVC's programming switch is set to the OFF position. **When attaching a ZEPROM, it is critical that the module is correctly attached. The module must be orientated such that the erasure window is uppermost.**

When the ZEPROM module is attached to the ZVC, and the programming switch is set to ON, the ZEPROM module is forced into a *straight 16K mode*. In this mode, all four 4K cores in the ZEPROM can be programmed. However, because the module appears as 16K, it will always consume the addressing space of *two ports* whilst it is being programmed. For this reason, the user **must** ensure that the port adjacent to the ZVC is empty, or contains only memory modules or system modules such as HP-IL, printer, Timer, etc. If this is not done, there will be an addressing conflict and data will not be correctly verified after burning.



Whilst data is being burnt into a module, you should avoid handling the ZVC, module or HP-41. If the PROGRAMMER ROM software is used, the HP-41's keyboard is disabled during burning; however, this may not be the case with other controlling software.

Batteries

Because the HP-41's own batteries are used by the ZVC to generate a 12.5-volt burning voltage needed to burn data into a ZEPROM module, it is essential that adequate charge is available in the batteries whenever burning is commenced.

Unless you will only be burning very small amounts of data, we recommend that you always start with a fresh set of batteries in your HP-41. When purchasing the batteries, buy them from shops that have a high turnover of such cells. While your local corner shop may be convenient, the batteries they stock may have been sitting on their shelves for quite a while.

Although it may also be tempting to use Ni-Cd cells, in most cases our advice is, **don't!** The re-chargeable battery pack manufactured by HP may not have the capacity to burn a 4K image into ZEPROM. This applies even if a mains recharger is connected, because the recharger cannot compensate for the drain on the cells. It is possible that using separate 'size N' Ni-Cd cells will suffice to power the burning of an image, but this cannot be guaranteed. For these reasons, we strongly recommend using alkaline batteries. With a fresh set of alkaline batteries, the user could expect to be able to burn in excess of 512K of data into ZEPROM modules.

The programming switch on the ZVC activates the voltage conversion. When ON, the 12.5V burning voltage is continually being generated, although the highest current drain only occurs when programming is actually taking place. To minimise the current drain from the cells, we recommend that you only turn the switch to ON just before beginning to burn and then turn it OFF again immediately after finishing. With the switch set to OFF, you can access (read) the module plugged onto the ZVC as if it were plugged directly into that port. The configuration that you have burnt into the module is also effective in this condition. Note that when the programming switch is set to ON, the module always appears to the HP-41 as if it were a straight 16K module.

Care and Maintenance

The ZVC does not contain any user serviceable parts and should not require servicing during normal or extended use. However, whilst not in use the user should keep the ZVC in the antistatic packaging in which it is supplied and protect it from physical damage.

Whenever the ZVC is handled, the user should avoid touching any contacts, components or the underneath of the printed circuit board. Try to hold it by gripping the PCB edges between thumb and forefinger. Over a long span of time, it is possible that the ZVC's contacts, where the ZEPROM module is plugged onto it, may become oxidised. If this should happen, very gently rub the contacts with a soft pencil eraser. Avoid using a hard, typewriter eraser as this can easily damage the contacts.

Chapter 5

Burning Methods

The ZEPROM module has been designed to be as flexible as possible within the constraints of the HP-41 system.

Burning (also called programming) of ZEPROMs, as with other EPROM type devices, involves the use of a programming fixture, onto which the ZEPROM is placed, and suitable software driving the fixture and controlling the burning process itself.

Burning Fixtures

In designing ZEPROM, the intention has been to provide solutions for differing markets:

- a cost effective solution that could be used by the HP-41 user without considerable technical expertise and investment in equipment. This is the *ZEPROM Voltage Converter* (ZVC); a simple device that plugs directly into one of the four I/O ports on the HP-41 and then allows the ZEPROM module to be plugged onto it. To use the ZVC, ancillary software must be used to control the burning process. The recommended software is the PROGRAMMER ROM that is pre-programmed into one ZEPROM core.
- a medium to high volume *ZEPROM Gang Burner* that allows up to eight ZEPROMs to be burnt and tested simultaneously. The device has built in software and is intended for connection to a microcomputer via RS232C. In addition, it can be used in a stand-alone mode to copy existing modules. The unit is recommended for use with various HP-41 software development systems available for most MS/PC-DOS desktop computers.

In addition to these fixtures designed and marketed by Zengrange Ltd, another solution is available from a third party source:

- The *ZEPROM Programmer* is available from Firmware Corporation, 605 NW 5th Street, Suite 2A,Corvallis, OR 97330, USA. This fixture is intended for low to medium volume use, and can burn two ZEPROMs simultaneously. The Zeprom Programmer can be connected by a cable to the HP-41, or via a RS232C connector, to a micro computer running suitable data-communication software. This product also contains additional RAM and controlling software allowing the user to build up and test the complete image before burning. The unit is recommended for use with a number of HP-41 Software Development Systems available for MS-DOS computers.

Controlling Software

In addition to dedicated software packaged inside some programming fixtures, the following may be used for developing and burning ROM images into ZEPROM.

The PROGRAMMER ROM

The PROGRAMMER ROM is a companion product for the ZEPROM Voltage Converter. Its software provides complete control over the burning of both user code (RPN) programs and M-Code (Machine language) functions into ZEPROM.

The PROGRAMMER software is burnt into each ZEPROM module as part of the extensive quality control procedures used during manufacture. This ensures that each user can immediately begin burning ZEPROM, without needing to purchase or write their own programming software, and also ensures that the correct ZEPROM burning algorithm is used. An added innovation, is the provision of functions allowing bank switching of user code programs. Until ZEPROM and the PROGRAMMER ROM, this was not possible. Also included in the function set are routines to copy code between an HP-IL mass storage device and ZEPROM.

Chapter 5 - The Programmer ROM, contains a complete description of all PROGRAMMER ROM functions.

HP-41 Software Development Systems

A number of *HP-41 Software Development Systems* have been released as complete development packages running on MS/PC-DOS and CP/M desktop computers. Most of the packages provide the facilities to develop user code and/or machine code routines in the format required by HP for burning into a mask-programmed ROM. As ZEPROM is completely downwards compatible with HP's ROM formats, these SDS packages can be used to develop code for and burn ZEPROM modules. These modules can then be used for field testing before submitting to HP for ROM burning.

ZENROM-3B - The Programmer's Module

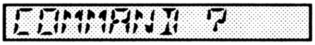
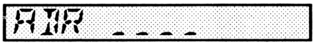
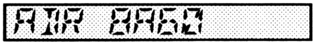
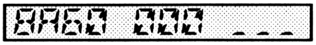
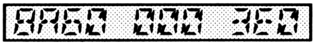
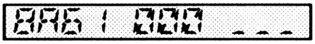
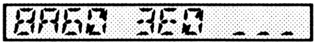
The ZENROM Programmer's Module, another Zengrange product, opened up the world of synthetic and machine code programming to the HP-41 user by providing unique functions that permit direct access to the HP-41 operating system.

Included in its function set is a machine code editor, MCED, that can be used to transfer code from a Q-ROM (Quasi-ROM) device to ZEPROM or to burn directly into a ZEPROM module. When using ZENROM to burn a module, it is essential that the correct burning algorithm is followed.

Example using ZENROM with the ZVC

This example illustrates using ZENROM to write the single machine code word 3E0h at location 8A60h. It assumes that you have already checked the HP-41's batteries, plugged the ZVC into port 8 or 9, connected the ZEPROM onto the ZVC, and have pressed the burn switch to ON.

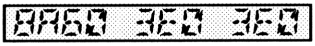
ZENROM functions can also be used to transfer blocks of code rather than single words. When doing this remember that you must verify correct burning, and when correct, repeat the burn again twice more to ensure ZEPROM stays programmed.

Key sequence & Description	Resulting Display
Execute the MCED function: XEQ ALPHA M C E D ALPHA	 COMMAND ?
MCED now waits for a command to be entered. Use the goto function to enable movement to an address: ■ GTO	 ADR _ _ _ _
Key in the address 8A60h: 8 A 6 0	 ADR 8A60
R/S	 8A60 000 _ _ _
The three 000h digits indicate that the address 8A60h is currently blank.	
Key in the word value that you wish to burn at this address; i.e. 3E0h: 3 E 0	 8A60 000 3E0
R/S	 8A61 000 _ _ _
Back step to the previous address in order to check that the correct value was burnt: ■ BST	 8A60 3E0 _ _ _
Because EPROMs need a longer burn cycle than RAM, you may find that the word has not been burnt. If this is the case, re-key the word 3E0h by repeating the last two steps until the word is correct.	

Once correct, you must re-burn the word *twice more*. Because EPROMs may revert if the burn cycle was not long enough, you must do this to ensure that the word will remain programmed.

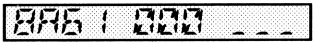
Key in the word value 3E0h again:

[3] [E] [0]



8A60 3E0 3E0

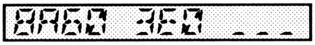
[R/S]



8A61 000 _ _ _

Back step to the previous address.

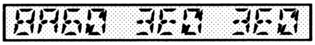
■ [BST]



8A60 3E0 _ _ _

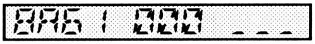
Key in the word value 3E0h again:

[3] [E] [0]



8A60 3E0 3E0

[R/S]



8A61 000 _ _ _

The word should now be programmed.

Other Software

Various software packages are available for writing code into Q-ROM or RAM Storage devices. Whilst not intended for burning ZEPROMs, they can often be used for that purpose with the ZVC. In most instances, it is necessary to write small user code routines using functions in the software. This is necessary because programming an EPROM requires a longer write cycle.

In general, software that uses the machine code WMLDL (040h) instruction for writing to Q-ROM devices, with the address in C[6:3] and the data in C[X], can be used with ZEPROM. However, it should be noted that a special algorithm should be used for programming the module as ZEPROM uses the Peripheral Flag 4 (accessed with the ?PF 4 (06Ch) instruction) to control burning. This algorithm is discussed in *Appendix A*.

Reference should be made to the relevant manuals or product manufacturers for further details of the programming methods and software. A typical example of this type of software is:

MLDL Operating System/David Assembler

The MLDL Operating System & David Assembler are products of the Dutch company Eramco Systems BV (Loodsgracht 23, 1781 KM Den Helder, The Netherlands). Both of these are machine code development utilities intended for use with Eramco's own RAM Storage Units (Q-ROMs). Many of the functions provided in these software packages can be incorporated into user code routines to control the burning of software into ZEPROM.

Chapter 6

The Programmer ROM

The PROGRAMMER ROM software has been designed and especially written to provide complete control over the burning of both User Code (Reverse Pólish Notation) programs and M-Code (Machine Code) functions into a ZEPROM. The software is intended as a companion product to the ZEPROM Voltage Converter (ZVC). When used together, they form a powerful, low-cost solution for low-volume burning of ZEPROM Modules. The simple, user orientated design of the complete package also brings the burning of ZEPROMs well within the ability of normal users who might want to have their favourite RPN programs permanently available in a module.

The software contains an instruction set of special machine coded functions that perform necessary housekeeping tasks and control burning at the byte, program, block and ROM page level. To aid the non-technical user code programmer, these functions have also been incorporated into an automated, prompting burn program. For simplicity, the software is described in two sections:

- *Instruction Set* describes all the low level functions giving ultimate control over each stage of burning;
- *Utility Routines* describes the user orientated routines that automate the burning process by prompting for input of relevant arguments.

These software descriptions assume the user understands the various HP-41 owner's manuals, is proficient in user code programming and conversant with the general terms and concepts of the HP-41 computer. To successfully use this software in user code programs, the user should also understand how flag 25 can be used to affect trapping and handling of burn and other errors.

For more detailed information about the 41's structure, ROM formats, machine code and synthetic programming, we recommend reading the ZENROM Programmer's Module handbook - which is also available from Zengrange Ltd.

!

©
Copyright

!

Caution

The PROGRAMMER ROM contains functions that facilitate the copying of ROM software. However, since software is covered by copy, design or patent rights, it is illegal to make copies for distribution, duplication or modification unless permission has been granted by the holder of those rights. Obtaining such permission is the specific responsibility of the user.

In cases where a Zengrange product is used to infringe copy, design or patent rights, the user shall bear the entire responsibility and assume all liability for infringement. Neither Zengrange Ltd, nor its agents, shall be in any way liable for the user's actions.

The Instruction Set

The PROGRAMMER ROM Instruction Set incorporates all those functions necessary for ultimate control over the burning process. This may be necessary for reasons of burning only a small number of bytes, or because the user wishes to burn ZEPROM in a particular way. A complete understanding of the instruction set is not necessary for users wishing to burn programs from the HP-41, or for copying modules or code from a Q-ROM/RAM storage device. However, we still recommend reading the complete manual, as this will give a better understanding of the procedures involved.

Instruction set functions are described in alphabetical order. Unless stated otherwise, functions can be executed from the keyboard, or under control of a running program.

Keying in Arguments for Instructions

In general, instructions requiring the user to supply arguments (parameters) for addresses, pages, number of bytes, etc., allow this to be done in two forms:

a floating point decimal value e.g. **40812.**	Decimal addresses range from 0 to 65535. However, since locations below 32768 are used by the HP-41 itself, valid input for all burning operations is limited to between 32768 & 65535. Similarly, valid pages for burning operations are from 8 to 15.
a hexadecimal alpha string e.g: **9F6C**	Hexadecimal addresses range from 0 to FFFFh. However, since locations below 8000h are used by the HP-41 itself, valid input for all burn operations is limited to between 8000h & FFFFh. Similarly, valid pages for burning operations are 8h to Fh.

Where permitted, the user may enter values in either format depending upon preference. Those functions that return a value to registers or the display, will generally use the same format as that of the input. However, the incrementing burn counter and burn error messages always display addresses in hexadecimal.

Hexadecimal values can be keyed into the Alpha register and then stored into a stack register by the ASTO function. E.g. ASTO . X would copy a string of up to six characters from Alpha into Reg X. Hexadecimal values can be entered in either upper or lower-case letters, e.g. '9f6a' or '9F6A', with valid input being 0-9, A-F and a-f. For clarity in this manual we have shown hexadecimal values in uppercase, followed by the letter 'h'; e.g. '9F6Ah'.

Two utility functions, HEXDEC and DECHEX, have also been provided for converting the Reg X content between the two formats.

Test Functions

In general, the test functions, those with names terminating in a '**?**', follow the usual HP convention and perform tests in the same manner as standard HP-41 test functions such as X=Y?, X<=Y?, etc. Different results are obtained depending upon whether the function is executed from the keyboard, or from a running program.

Under Keyboard Execution

When executed from the keyboard, these functions display a **YES** or **NO** answer depending upon the outcome of the test:

If test was TRUE, display:	**YES**
If test was FALSE, display:	**NO**

Under Program Control

Under program control there is no displayed result, but the program instruction following the test will be executed or skipped depending upon the test. The **"DO IF TRUE"** rule is followed by all these tests:

If test was TRUE:	**PERFORM the next program step**.
If test was FALSE:	**SKIP the next program step**.

Some functions may also modify the content of stack registers dependent upon the outcome of the test.

Bank Switching Instructions

Special bank switching instructions are provided by the PROGRAMMER ROM to allow the user to more fully utilise a bank-switched ZEPROM. These instructions allow the user to branch to and execute user code subroutines in other banks, and to swap the banks from the keyboard. The instructions, BGTOxx, BRTNxx, BXEQxx, PBxx and SBxx, are loaded into bank-switched ZEPROM banks by the PROGRAMMER ROM functions ADDUCF and ADDMCF. During loading, the bank-switching instruction always take on the XROM identity of the ZEPROM bank into which they are being loaded. E.g. if a bank is initialised with an XROM identity of 28, the functions would appear as [BGTO28], [BRTN28], [BXEQ28], [PB28] and [SB28].

The reason for this adoption of the XROM identity is that such functions can only operate on and within the particular bank in which they reside. Because each ROM or ZEPROM page connected to the 41 must have a unique XROM identity, the instruction itself must possess that same XROM identity. In describing the functions in this manual, we have referred to them as [BGTOxx], [BRTNxx], [BXEQxx], [PBxx] and [SBxx].

When writing user code programs for ZEPROM (that will call subroutines in other banks of a bank-switched ZEPROM), the user **must** ensure that the instruction has the correct XROM identity; this always being that of the ZEPROM bank into which the program is to be loaded. For example, if a program in one bank (XROM 28) is to execute a subroutine in a switched bank (XROM 15), the programs would be written as follows:

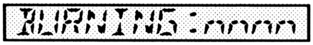
```
        Main Program                  program            Sub-Program
     (in bank of XROM 28)              flow          (in bank of XROM 15)

♦LBL "MAIN"                                        :
 :                                                 :
 :                                                 :
 "SUB"        Name of sub-prog          ┌►  ♦LBL "SUB"    Entry from main
 ASTO X       Store name in Reg X       │    :            prog in other bank
 BXEQ28       Enable other bank        ─┘    :
 :            Return from sub-          ◄┐   :
 :              program                  │   :
 :                                       └   BRTN15       Return to main
 :                                            :            prog in other bank
 :                                            :
```

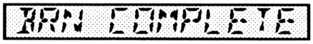
Burn Instructions and Output Formats

All PROGRAMMER ROM instructions to burn data into ZEPROM pages have a common display and output format. During burning, the display will show one of the following message sequences:

Successful Burn

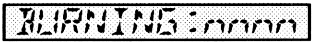

BURNING:nnnn

followed by:

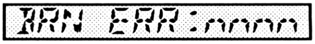

BRN COMPLETE

a TONE 7 is also sounded to indicate completion.

Unsuccessful Burn


BURNING:nnnn

followed by:


BRN ERR:nnnn

a TONE 0 is also sounded to indicate the error.

Errors during burning operations generally occur when the PROGRAMMER ROM reads back and compares the information that it has just tried to burn into ZEPROM. If the data is not identical, the PROGRAMMER ROM considers this to be a burn error and so aborts burning. In most instances, such burn errors are caused by low HP-41 batteries, attempting to burn into a non-ZEPROM module page or an already burnt page, or not having the ZVC programming switch set to the ON position.

The PROGRAMMER ROM's burn instructions fall into two groups:

Burning User Code Programs

BURNUC provides the necessary control to burn user code programs from main memory into a ZEPROM page. During burning, stack registers are restored or modified as follows:

During user code burning:

Reg X: contains the address of the first line of code burnt.

User code burn succeeded:

On completion, stack registers are restored to their original content.

Reg X: page number to burn into.

Alpha: name of program to burn.

Burn operation failed:

Reg X: replaced by the burn failure address. This is the address at which the data it tried to burn was verified incorrectly.

Reg L: contains the address of the first line of code burnt.

Alpha: name of main memory program to be burnt into ZEPROM.

If an error should occur whilst burning a user code program, it is essential that the user **does not** just re-try using BURNUC. If restarted, BURNUC will insert the new image after that of the aborted attempt. To prevent this, a REBURN instruction has been provided for recovering from user code burn errors. Please note that REBURN **must not** be used in other instances.

As input REBURN requires:

Reg X: address of the first line of the program code burnt. This is the start of the program that BURNUC was burning when error occurred.

Alpha: name of program BURNUC failed to completely burn.

Because the BURNUC burn error moves the start address to Reg L, the user **must** use LASTX to recall this to Reg X **before** using REBURN.

All Other Burning Operations

Various other instructions are provided in the PROGRAMMER ROM to control the burning of ZEPROMs at the page, block and word level. The input to these instructions varies according to the number of parameters required.

The stack registers are restored or modified by the instructions as follows:

Burn operation succeeded:

All stack registers: data is unchanged. The exact data content depends upon the actual operation currently being performed.

Burn operation failed:

Reg X: replaced by the burn failure address . This is the address at which the data it tried to burn was verified incorrectly.

Reg L: replaced by the original content of Reg X at the start of the operation.

Other registers: unchanged.

In the event of an error terminating the burning process, e.g. because of low batteries, the operation can usually be retried using the same data. However, because the value originally in Reg X will have been moved to Reg L, the user must recall this to Reg X before re-trying the instruction. Because these operations return the burn failure address to Reg X, this can be used with a suitable editor, such as ZENROM's MCED, to examine that address location in the ZEPROM page.

Function Description Index

ADDBSW Add bank-switching code to initialised page

XROM 09,01

Purpose Makes a blank or previously initialised page into bank switching. The function sets the bank switching bit at page location FFDh and adds the necessary bank switching code in the locations FC7h to FCAh - these being the locations reserved for that purpose by Hewlett-Packard.

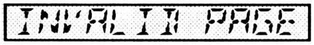
Cautions ADDBSW requires that locations FC7h to FCAh and FFDh are currently unused. If subsequently intending to add the M-code bank switching functions PBxx and SBxx (added with ADDMCF) and/or the user code functions BGTOxx, BRTNxx and BXEQxx (added with ADDUCF), then the page locations FCBh through FF2h should also be unused.

Input The user must supply a value for the following argument:

Reg X:	*target-page*	Target page specified either as a floating point value, or a hexadecimal string.

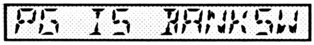
Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.



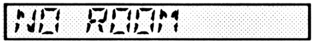
INVALID PAGE

An invalid page was specified.



PG IS BANKSW

Specified ROM page has already been initialised as being bank-switched.



NO ROOM

The page locations FC7h to FCAh and FFDh have already been used.

Example ADDBSW adds just the essential bank-switching code to a blank page, or one previously initialised with INITPG as being not bank-switched. It can therefore be used in setting up a bank-switched ZEPROM.

When initialising ZEPROM as a bank-switched module, *all* banks must be initialised as being bank-switched. Because it is not always practical to initialise all pages at once, ADDBSW can be used to add the necessary switching code into the other pages, until you are ready to INITPG all remaining banks with XROM-id, header, FAT, etc.. See also INITPG.

As described on page 16, ADDBSW can be used to add the necessary switching code into a blank alternate bank where a small number of large M-code functions are to be inserted into ZEPROM. In this way, valuable space in the secondary bank is not consumed by the XROM-id, header, FAT, etc..

ADDMCF — Add M-code bank-switching functions to initialised page

XROM 09,02

Purpose Adds the machine coded bank switching functions PBxx and SBxx into a page previously initialised as being bank-switched. The two instructions are loaded into the specified page at locations FE9h through FF2h and their names entered into the FAT. These instructions permit the user to switch between primary and secondary banks as required. During loading, these instructions always take on the XROM identity of that bank. E.g. if a bank is initialised with an XROM of 28, the functions would appear as PB28 and SB28. See page 39 for a more detailed description of this procedure.

Cautions ADDMCF requires that locations FE9h to FF2h are currently unused. If subsequently intending to add the user code instructions BGTOxx, BRTNxx and BXEQxx with ADDUCF, then the page locations FCBh through FE8h should also be unused.

Note that ADDUCF <u>must</u> be used to insert the bank-switching functions into both banks of that page. Failure to do so will mean that a particular bank can be enabled, but not disabled.

Input The user must supply a value for the following argument:

Reg X:	*target-page*	Target page specified either as a floating point value, or a hexadecimal string.

Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

INVALID PAGE	An invalid page was specified.
NOT SWITCHED	The page specified is not bank-switched.
FAT OVERFLOW	There are not enough spare FAT entries remaining.
INITPG FIRST	The page must be initialised before functions can be added into it.

ADDUCF — Add user code bank-switching instructions into initialised page

XROM 09,03

Purpose Adds the user code bank-switching functions BGTOxx, BRTNxx and BXEQxx into a page previously initialised as being bank-switched. The three instructions are loaded into the specified page at locations FCBh through FE8h and their names entered into the FAT. These functions permit the user to perform a GTO, XEQ and RTN between running user code programs in different banks of a switched ZEPROM. During loading the instructions always take on the XROM identity of that bank. E.g. if a bank is initialised with an XROM of 28, the functions would appear as BGTO28, BRTN28 and BXEQ28. See page 39 for a more detailed description of this procedure.

Cautions ADDUCF requires that locations FCBh to FE8h are currently unused. If subsequently intending to add the M-code instructions PBxx, and SBxx into the bank with ADDMCF, then the locations FE9h through FF2h should also be unused.

Note that ADDUCF <u>must</u> be used to insert the bank-switching functions into both banks of that page. Failure to do so will mean that a particular bank can be enabled, but not disabled.

Input The user must supply a value for the following argument:

Reg X:	*target-page*	Target page specified either as a floating point value, or a hexadecimal string.

Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

INVALID PAGE	An invalid page was specified.
NOT SWITCHED	The page specified is not bank-switched.
FAT OVERFLOW	There are not enough spare FAT entries remaining.
INITPG FIRST	The page must be initialised before functions can be added into it.

BGTOxx User code bank-switching GTO instruction in ROM with XROM-id of xx

Purpose Perform a bank-switched GTO instruction to a user code program or subroutine in an alternate bank.
When the HP-41 encounters a normal GTO instruction in a running program, it searches both RAM and ROM for the specified LBL and then transfers execution to that program. However, with bank switched ROMs, the 41 cannot see programs in alternate banks unless those banks are first enabled. For this reason a special GTO function has been included to perform the bank switching. The function, BGTOxx, being loaded into a switched bank by ADDUCF. During loading the instruction always takes on the XROM identity of that bank. See page 39 for a fuller explanation.

Action When BGTOxx is encountered in a running program, in place of the normal GTO instruction, the HP-41 switches banks in that page and uses a GTO IND X sequence to search for and branch to the program whose name the user has stored as an alpha string in Reg X.

Cautions **This instruction should be used with extreme care as no error checking can be performed by the function**. Users should ensure that the specified global LBL actually exists in the alternate bank and that the alternate bank routine has been written for bank-switched usage. For example, unless a BRTNxx is used to replace a terminating RTN, the program will terminate with the alternate bank still enabled.
Because ZEPROM always switches both banks of both pages at the same time, the subroutine can reside in either page, but must be in the alternate bank and in the same ROM.
In writing programs to branch to routines in another bank, the user must replace GTOs with BGTOxx instructions wherever bank-switching is required and ensure that the XROM identity; is used.
Because BGTOxx takes its LBL as an alpha string from Reg X, a maximum of 6 characters can be used for the label.

Input The user must supply a value for the following argument:

Reg X: *alpha LBL* The global alpha LBL to be called stored as alpha string. Maximum string length is 6 characters.

Output Stack remains unchanged.

Reg X: *alpha LBL* The global alpha LBL to be called stored as alpha string. Maximum string length is 6 characters.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

NONEXISTENT

A non-existent global alpha LBL was specified in Reg X.

Example

This example illustrates the use of BGTOxx to branch execution of a program running in the primary bank to a subroutine in the secondary bank. The primary bank has been initialised with the XROM identity 21 and the secondary bank with XROM 31.

Primary bank code (XROM 21)

```
♦LBL "PROG1"        Main program in primary bank.
 :                  :
 :                  :
 "PROG2"            Name of program to execute
 ASTO X             Load alpha string into Reg X
 BGTO21             Go to the label in the other bank
 :                  Program counter is left here after
                        returning from secondary bank
```

Secondary bank code (XROM 31)

```
♦LBL PROG2          Program in secondary bank
 :                  :
 :                  :
 BRTN31             End subroutine in secondary bank,
                        swap back to primary bank
```

BLANK? — Test if the page in Reg X is blank

XROM 09,04

Purpose Checks every location in the 4K page specified in Reg X and verifies that it is blank. The function behaves as a normal HP-41 test function.

Input The user must supply a value for the following argument:

Reg X:	*target-page*	Page to verify specified as a floating point value, or hexadecimal string.

Output The function's action varies depending upon it being executed from the keyboard, or under program control. See Page 39 for details of the general format of comparison tests.

Under Keyboard Execution:	Displays a YES or NO answer. **Reg X:** unchanged (*target-page*)
Page is blank (TRUE)	Displays 'YES' answer.
Page is not blank (FALSE)	Displays 'NO' answer.
Under Program Control:	Doesn't display an answer, rather executes or skips the next program instruction depending upon the result. Follows 'DO IF TRUE' rule:
Page is blank (TRUE)	Executes the next program step. **Reg X:** unchanged (*target-page*)
Page is not blank (FALSE)	Skips the next program step. **Reg X:** unchanged (*target-page*)

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

INVALID PAGE

An invalid page was specified.

Example This example uses BLANK? to creates in Alpha a string of hexadecimal characters representing all blank pages found between 8h and Fh.

```
01♦LBL "BLANKS"      Program name
02 CLA               Clear alpha ready for list
03 8.015             Load ISG control number (8-15)
04♦LBL 00            Start of ISG loop
05 BLANK?            Is this page blank ?
06 XEQ 01            Yes, call 'append' subroutine
07 ISG X             Increment page number
08 GTO 00            Repeat to check next page
09 RTN               RTN when all pages checked
10♦LBL 01            Start of 'append' subroutine
11 DECHEX            Convert page number into hex
12 ARCL X            Append page number to Alpha
13 X<> L             Restore ISG control to Reg X
14 RTN               End of subroutine
```

BNKSW? Test if page in Reg X is bank-switched

XROM 09,05

Purpose Tests to see if the page specified in Reg X is bank-switched. The bank-switching protocol developed by Hewlett-Packard requires that a bank-switched page has a non-zero value in the most significant two bits of the word at page location FFDh. BNKSW? therefore checks this location and returns an indicator value to Reg X. Because ZEPROM uses a superset of HP's protocol, BNKSW? is able to distinguish between pages in an HP ROM, and primary and secondary banks of a ZEPROM. With HP switched banks, the value returned is '-1'. For ZEPROMs, a value of '+1' or '+2' is returned indicating primary (bank 1) or secondary (bank 2) respectively.

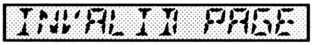
Input The user must supply a value for the following argument:

Reg X:	*target-page*	Page specified as a floating point value, or a hexadecimal string.

Output The function's action depends upon keyboard or program execution. See Page 39 for details of the general format of comparison tests.

Under Keyboard Execution:	Displays a YES or NO answer. **Reg X:** is replaced by the bank switched status value indicator **Reg L:** contains the target page originally in Reg X.
Page is switched (TRUE)	Displays 'YES' answer. **Reg X:** contains either: -1 = HP primary/secondary bank +1 = ZEPROM primary bank +2 = ZEPROM secondary bank
Page not switched (FALSE)	Displays 'NO' answer. **Reg X:** contains the value '0'.
Under Program Control:	Doesn't display an answer, rather executes or skips the next program step depending upon the result. **Reg X:** is replaced by the bank switched status indicator in FFDh. **Reg L:** contains the target page originally in Reg X.
Page is switched (TRUE)	Performs the next program step. **Reg X:** contains either: -1 = HP primary/secondary bank +1 = ZEPROM primary bank +2 = ZEPROM secondary bank
Page not switched (FALSE)	Skips the next program step. **Reg X:** contains the value '0'

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.



INVALID PAGE

An invalid page was specified.

Example This example uses **BNKSW?** to determine if the page specified in Reg X is a bank-switched ZEPROM page. If unswitched, or in an HP bank-switched ROM, it displays a message and exits. If a bank-switched ZEPROM page, the enabled and disabled banks are swapped over. The program expects the number of the page to be tested in Reg X.

```
01♦LBL "SWPBNK"       Program: Swap ZEPROM banks.
                         Expects the page # in Reg X
02 BNKSW?             Is page bank-switched?
03 GTO 00             Yes, so go check what type
04 "NOT SWITCHED"     No, so load error message, then
05 GTO 99             Go to display routine
06♦LBL 00             Check bank switched type
07 X>0?               Is page a ZEPROM? (value=1 or 2)
08 GTO 03             Yes, so go swap banks
09 "HP SWITCHED"      No, so load "HP" message
10♦LBL 99             Display routine entry point
11 AVIEW              View message in Alpha register
12 RTN                Exit
13♦LBL 03             ROM is ZEPROM & bank-switched
14 X<> L              Put page # in Reg X and current
                         bank in Reg L
15 GTO IND L          Branch depending upon currently
                         enabled bank
16♦LBL 01             Primary bank currently enabled
17 ENABLES            So enable secondary bank
18 RTN                Exit
19♦LBL 02             Secondary bank currently enabled
20 ENABLEP            So enable primary bank
21 RTN                Exit
```

BRTNxx User code bank-switching RTN instruction in ROM with XROM-id of xx

Purpose Perform a bank-switched RTN instruction to a user code program or sub-routine in an alternate bank.

Theory When the HP-41 encounters a normal RTN instruction in a running program, it takes the next pending return address and transfers execution to that address. However, if called as a sub-routine by a program in another bank of a bank-switched ROM, that address will be incorrect because it resides in a different bank. For this reason a special Return instruction, BRTNxx, has been provided to perform the bank switching. During loading with ADDUCF, the function always takes on that bank's XROM identity. See page 39 for a full explanation of the procedure.

BRTNxx instructions should replace RTNs wherever bank switching is required. However, if the routine is to be called by programs within that bank **and** from other banks, then the user must ensure that the correct type of return is used. The most suitable method of doing this is to set or clear a flag depending upon the calling program. By testing the flag, either a RTN or BRTNxx can be executed. BRTNxx should also be used to precede the normal END if that instruction is also to bank-switch execution back to a calling program.

Action When BRTNxx is used in place of the normal RTN instruction, the banks are switched and then execution continues from the next pending RTN address. If no return is pending, execution will halt after the banks have been swapped.

Cautions **This instruction should be used with extreme care. Because no error checking can be performed, users must ensure that BRTNxx is only used when it is necessary to return control to another bank.** If incorrectly used (instead of a RTN), the banks will swap, and program execution will continue from the next address on the return stack. This could contain user or machine code, so results are totally unpredictable!

Input None.

Output None.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

Example This illustrates the use of BRTNxx to bank-switch return from a sub-routine in another bank. The routine also checks to determine whether it was called from the same bank; in which case a normal RTN is used. The primary bank has been initialised with the XROM identity 21 and the secondary bank with XROM 31.

Primary bank code (XROM 21)

```
♦LBL "PROG1"      Main program in primary bank.
 :                :
 :                :
 "PROG3"          Load alpha LBL
 ASTO X           Load as alpha string into Reg X
 SF 0             Set flag = IS a bank-switched call
 BXEQ21           Call sub-routine label specified
 :
 END              End of main program
```

Secondary bank code (XROM 31)

```
♦LBL PROG2        Program in secondary bank
 :                :
 :                :
 CF 0             Clear flag = IS NOT bank-switched
 XEQ "PROG3"      Call subroutine in this bank
 :                :
 :                :
 END              End main program in this bank
 :                :
 :                :
♦LBL "PROG3"      Start of subroutine
 :                :
 :                :
 ?FS 0            Called by another bank?
 BRTN31           Yes, do switched return to primary
 RTN              No, do an ordinary return
```

BURNUC Burn user code program from RAM or ROM

XROM 09,06

Purpose Takes a user code (RPN) program from main memory, another plug-in module, or a Q-ROM/RAM Storage Device and burns it into the ZEPROM page specified in Reg X. The function requires a program name in Alpha and searches sequentially through RAM followed by plug-in's, and uses the first program it finds that contains a global alpha LBL of that name. If alpha is empty, **BURNUC** takes the program to which the program pointer is currently located; provided that it contains has at least one global alpha LBL.

BURNUC requires that the program to be loaded has already been packed. If not already packed, use PACK or press **GTO** **.** **.** before executing **BURNUC**. If the program came from RAM **BURNUC** will compile it (to link XEQs and GTOs with their labels) before burning the program into the specified page. Programs taken from ROM are deemed to already be compiled, so **BURNUC** does not do so. When the program is loaded, a FAT entry is made for every global LBL in the program.

If burn errors occur due to low batteries, etc., the burn failure address is returned to Reg X. The **REBURN** instruction can be used to recover and complete the burn operation, but the user must recall the burn starting address to Reg X from LASTX. See page 41 regarding use of **REBURN**.

Cautions Enough spare FAT entries must exist for every global alpha label in the program to be burnt.

BURNUC does not search for program files in Extended Memory.

If your program contains synthetic instructions (and intentional nulls) you may not want it to be packed. In this case you must **not** use **BURNUC**. Instead, execute **FRSPC?** to ascertain the amount and address of free space in a specified page, then execute the **REBURN** instruction to burn the program into the page. **REBURN** is used in this case to bypass the compilation and packed status checks in **BURNUC**.

Input The user must supply values for the following arguments:

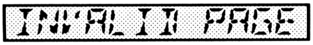
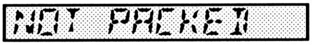
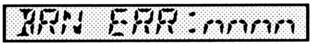
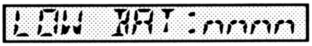
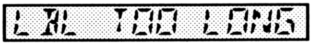
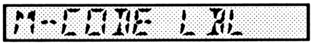
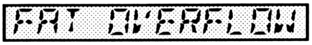
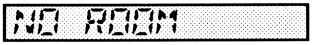
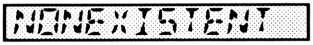
Reg X:	*target-page*	Page specified as either a floating point value, or a hexadecimal string.
Alpha:	*program-LBL*	Maximum 7 character global alpha LBL. If blank takes program to which the pointer is positioned.

Output Stack registers are unchanged unless a burn error occurs. See page 40 for standard outputs. In the case of burn errors, registers are modified to:

Burn operation failed:	**Reg X:** replaced by the burn failure address (address at which the data was verified incorrectly).
	Reg L: contains the address of the first line of code burnt.
	Alpha: *program-name* being burnt.

Errors

See the *HP-41 Owner's Manuals* for standard messages. BURNUC will also return messages in case of compilation errors. See COMPUC for details.

 INVALID PAGE	An invalid page was specified.
 NOT PACKED	The specified user code program has not been packed.
 BRN ERR:nnnn	A burn error occurred at address indicated by '*nnnn*'. Commonly caused by specifying an empty or non-ZEPROM destination page.
 LOW BAT:nnnn	Low battery terminated burning at the address 'nnnn'. Check, replace the batteries and then REBURN.
 LBL TOO LONG	The program LBL in Alpha exceeds 7 characters.
 M-CODE LBL	The program LBL in Alpha is that of a machine code function in ROM or in the HP-41 operating system.
 FAT OVERFLOW	Not enough spare FAT entries for the number of global alpha LBLs in specified program. Change some to local LBLs.
 NO ROOM	Not enough free space is left to load the program specified in Reg X.
 NONEXISTENT	A non-existent global alpha LBL was specified in Alpha.

BURNWD — Burn word in Reg X into address in Reg Y

XROM 09,07

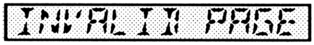
Purpose Burns an individual 10-bit HP-41 word specified in Reg X into the 16-bit ROM page address that is specified in Reg Y.
The input required for BURNWD is compatible with the output of CHKSUM so that BURNWD can be used to burn a checksum into the page after all other data has been burned.

Input The user must supply values for the following arguments:

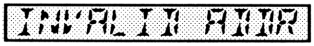
Reg Y:	*target-address*	Target address input as a floating point value, or hexadecimal string.
Reg X:	*10-bit-word*	Word to burn input as a floating point value, or hexadecimal string.

Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of standard outputs following burn errors.

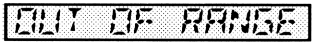
Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.


INVALID PAGE

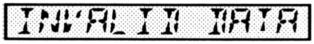
An invalid page was specified.


INVALID ADDR

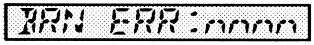
A negative or other invalid address value was specified in Reg Y. Check that valid and positive hexadecimal characters were used.


OUT OF RANGE

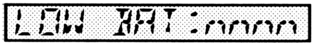
The checksum value in Reg X exceeds '3FFh' (1023).


INVALID DATA

A negative or other invalid data value was specified in Reg X.


BRN ERR:nnnn

A burn error occurred at the page address '*nnnn*'.


LOW BAT:nnnn

Low battery terminated burning at address location 'nnnn'. Check and replace batteries.

Example BURNWD is used to burn a computed page checksum into location FFFh. The checksum is computed by CHKSUM, which returns a checksum and its address to Reg X & Reg Y respectively. Program expects a page number in Reg X and exits with the checksum read by READWD in Reg X.

```
01♦LBL "BRNSUM"        Burn computed page checksum.
                          Expects page number in Reg X
02 CHKSUM              Get checksum & address into X & Y
03 BURNWD              Burn that value into ROM page
04 RDN                 Move address to Reg X
04 READWD              Read word at that address in ROM
04 RTN                 Exit
```

BXEQxx User code bank-switching XEQ instruction in ROM with XROM-id of xx

Purpose Perform a bank-switched execute (XEQ) instruction to a user code program or subroutine in another bank
When the HP-41 encounters a normal XEQ instruction in a running program, it searches both RAM and ROM for the specified LBL and then transfers execution to that program. However, with bank switched ROMs, the 41 cannot see programs in alternate ROM banks unless those banks are first enabled. For this reason, a special Execute instruction, BXEQxx, being provided to perform the bank switching. During loading into a switched bank by ADDUCF, the instruction always takes on the XROM identity of that bank. See page 39 for a fuller explanation.

Action Whenever BXEQxx is encountered in place of a normal XEQ instruction, the HP-41 saves the RTN address in the current bank, switches ROM banks and uses the XEQ IND X sequence to search for and execute the function whose name the user has stored as an alpha string in Reg X.
With the program name as an alpha string in Reg X, it may also be used from the keyboard to execute a program in the alternate bank.

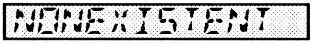
Cautions **This instruction should be used with extreme care as no error checking can be performed by BXEQxx.** Users **must** ensure that the called LBL actually exists in the alternate bank and that the called routine has been written for bank switching. Wherever the called routine returns to the calling routine, the RTN must be replaced with a BRTNxx. If a normal RTN or END is used instead, the 41 will not swap banks, but simply use the next return stack address and continue execution in the current bank. Depending upon the particular ROM, this address may contain user or machine code, so the result is totally unpredictable.
Because ZEPROM always switches both banks of both pages at the same time, the called sub-routine can reside in either page, but must be in the alternate bank and in the same ROM.
In writing programs to call subroutines in another bank, the user must replace XEQs with BXEQxx instructions wherever bank switching is required and ensure that the correct XROM identity is used.
Because BXEQxx takes its LBL as an alpha string from Reg X, a maximum of 6 characters can be used for the alpha label.

Input The user must supply a value for the following argument:

Reg X:	*alpha LBL*	The global alpha LBL to be called stored as an alpha string. Maximum strii ig length is 6 characters.

Output None.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.


NONEXISTENT

A non-existent global alpha LBL was specified in Reg X.

Example

This illustrates how to call a sub-routine in either the primary or secondary bank of the module. The primary bank is currently enabled and has an identity XROM of 21. Note that for the error trapping, via flag 25, to work correctly, the program expects the primary bank routine to leave flag 25 set.

Primary bank code (XROM 21)

```
01♦LBL "FNDXEQ"        Program: Find & Execute. Expects
                           program name to be in alpha
02 ASTO X              Store program name in Reg X
03 SF 25               Ignore error if occurs
04 XEQ IND X           Try to find LBL in this bank (NB. the
                           routine should leave Flag 25 set)
05 ?FS 25              Was LBL found (F-25 still set ?)
06 RTN                 Yes, thus do normal RTN to caller
07 BXEQ21              No, so swap banks and search for it
                           in other bank. Halt if not found
08 RTN                 Sub-routine found and executed
```

CHKSUM — Compute & return checksum

XROM 09,08 — for specified page

Purpose Computes a checksum for any HP-41 page specified in Reg X. After calculation, that checksum along with its address are returned into Reg X and Reg Y respectively ready for burning into the page with **BURNWD**.

Input The user must supply a value for the following argument:

Reg X:	*target-page*	Target page specified as a floating point value, or hexadecimal string. Valid pages are 0 to Fh, 0 to 15.

Output

For a valid page number:

Reg Y:	*checksum-address*	Checksum address in either floating point or hexadecimal string format.
Reg X:	*page checksum*	Checksum in either floating point or hexadecimal string format.
Reg L:	*target-page*	Target page specified as a floating point value, or hexadecimal string.

For an invalid page number:

Reg X:	*target-page*	Target page specified as a floating point value, or hexadecimal string.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

INVALID PAGE	An invalid page was specified.

Example Uses **CHKSUM** to calculate the checksum of the page specified in Reg X and then verifies that this is the same as that already burnt into the page.

```
01♦LBL "VERSUM"      Program: Verify page checksum
02 CHKSUM            Get checksum & address
03 X<>Y              Move address to Reg X
04 READWD            Read that address in the ROM
05 X<>Y              Move actual checksum back to Y
06 RDN               X=actual, Y=correct checksum
07 "GOOD "           Load default "good" message
08 X≠Y?              Is actual checksum bad ?
09 "BAD "            Yes, so load "bad" message
10♦LBL 01            Display entry point
11 "├CHKSUM          Append "checksum" to message
12 AVIEW             Show message to user
13 RTN               End of routine
```

COMPUC Compile user code program in main memory

XROM 09,09

Purpose

Compiles and links all local and numeric GTO and XEQ instructions to their corresponding LBLs in the user code program specified in the Alpha register. By compiling programs, the effective execution speed is increased because the HP-41 needs to perform less label searching.

Theory

Whenever a program in RAM is run for the first time, the HP-41 has to find all LBLs in the program by searching:

- For numeric labels (0 to 99), or local alpha labels (A to J, a to e), the HP-41 attaches the actual GTO/XEQ jump distance to that LBL and therefore does not need to search again.
- For global alpha labels such as "AB99", the HP-41 must search *all* alpha LBLs in program memory and plug-in's to find that LBL *every time* the program is run. Searching always begins at the *last* LBL in memory and continues upwards to the first corresponding LBL is found.

ROM programs, however, **must always be compiled**, because the HP-41 cannot search ROM for numeric LBLs over 14 or any local alpha LBLs. If a program was not compiled before burning into ROM, the HP-41 will crash!

Action

COMPUC searches sequentially through all programs in main memory for the first program containing the program LBL specified in the Alpha register. If no program LBL was specified, COMPUC compiles the program to which the program pointer is currently positioned.

COMPUC requires that the program is already packed (removing null bytes). Use PACK or press GTO . . to pack the program.

Compilation aborts with a "NONEXISTENT" message if the sought local or numeric LBL is not available. At the same time the decimal value of that LBL is returned to Reg X.

Cautions

Programs for loading into ZEPROM must always be compiled - see Theory above. COMPUC expects the program to reside in main memory. If the program is subsequently to be loaded into ZEPROM, it must contain at least one global alpha LBL in order to appear in the FAT of that page.

Input

The user should supply a value for the following argument:

Alpha:	*program-LBL*	Global alpha LBL in the user code program to be compiled. If left blank, COMPUC takes the program in which the pointer is positioned.

Output

Registers are unchanged unless a compile error occurs.

Alpha:	*program-LBL*	Global alpha LBL within program being compiled, or blank if no LBL was specified.
Reg L:	*previous-Reg-X-value*	Previous value moved from Reg X.

Reg X: *LBL value* — Decimal value of nonexistent local alpha or numeric LBL. These values are shown in the following table:

LBL 00 00	LBL 01 01	LBL 02 02	LBL 03 03
:	:	:	:
LBL 100 100	LBL 101 101	LBL A 102	LBL B 103
LBL C 104	LBL D 105	LBL E 106	LBL F 107
LBL G 108	LBL H 109	LBL I 110	LBL J 111
LBL T 112	LBL Z 113	LBL Y 114	LBL X 115
LBL L 116	LBL M 117	LBL N 118	LBL O 119
LBL P 120	LBL Q 121	LBL⊦ (R) 122	LBL a 123
LBL b 124	LBL c 125	LBL d 126	LBL e 127

(N.B. Labels 100, 101, T, Z, Y, X, L, M, N, O, P, Q, ⊦ (R) can only be created by means of synthetic programming techniques (S.P.))

Numeric and Local Alpha Label Values
(decimal values returned by [COMPUC] to Reg X)

[Errors] Refer to the *HP-41 Owner's Manuals* for details of standard messages.

NONEXISTENT — A non-existent global alpha LBL was specified in Alpha.

The program contains a GTO or XEQ to a non-existent local LBL.

NOT PACKED — The specified user code program has not been packed.

M-CODE LBL — The program LBL in Alpha is that of a machine code function in ROM or in the HP-41 operating system.

COPYPG — Copy entire 4K image between pages X & Y

XROM 09,10

Purpose Copies a complete 4K ROM image from an origin page specified in Reg Y into a destination page specified in Reg X. This function is useful for copying a complete block of code in a Q-ROM or RAM Storage Device, or for duplicating existing ROMs.

Cautions **Because software is covered by copy, design or patent rights, it is illegal to make copies for distribution, duplication or modification unless permission has been granted by the holder of those rights.** Obtaining such permission is the specific and sole responsibility of the user. In all cases where a Zengrange product is used to infringe such rights, the user shall bear the entire responsibility and assume all liability for infringement. Neither Zengrange Ltd, nor its agents, shall be in any way liable for the user's actions.

[COPYPG] is not able to detect the case where the pages specified in both Rg X & Y are those of an empty port. Because the burnt and read data are blank, [COPYPG] will believe that data was burnt correctly.

Input The user must supply values for the following arguments:

Reg Y:	*origin-page*	Origin page specified either as a floating point value, or a hexadecimal string.
Reg X:	*destination-pages*	Target page specified either as a floating point value, or a hexadecimal string.

Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

 INVALID PAGE	An invalid page was specified.
 INVALID X=Y	The origin and destination are identical.
 LOW BAT:nnnn	A low battery has caused burning to terminate at the address location indicated by '*nnnn*'. Check & replace batteries.
 BRN ERR:nnnn	A burn error occurred at the address indicated by '*nnnn*'. This is often caused by specifying a blank or non-ZEPROM destination page.

CPXYZ

Copy data as specified in Regs X,Y,Z

XROM 09,11

Purpose Takes the number of data bytes specified in Reg X from the origin address specified in Reg Z and copies it into the destination address specified in Reg Y. The maximum number of data bytes that can be specified is 4096; this being an entire 4K ROM page.

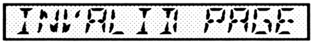
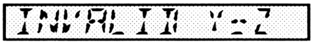
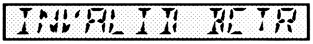
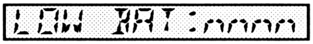
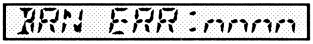
Cautions **Because software is covered by copy, design or patent rights, it is illegal to make copies for distribution, duplication or modification unless permission has been granted by the holder of those rights.** Obtaining such permission is the specific and sole responsibility of the user. In all cases where a Zengrange product is used to infringe such rights, the user shall bear the entire responsibility and assume all liability for infringement. Neither Zengrange Ltd, nor its agents, shall be in any way liable for the user's actions.

Input The user must supply values for the following arguments:

Reg X:	*number-of-bytes*	Number of bytes to copy beginning from the origin address. Can be specified as a floating point value, or a hexadecimal string. Maximum size is 4096 bytes (1000h).
Reg Y:	*target-address*	Target address specified as floating point value, or hexadecimal string.
Reg Z:	*origin-address*	Origin address specified as floating point value, or hexadecimal string.

Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

 INVALID PAGE	An invalid page was specified.
 INVALID Y=Z	The origin and destination are identical.
 INVALID BCTR	The byte counter in Reg X exceeds 4096 (1000h) or is negative.
 LOW BAT:nnnn	A low battery has caused burning to terminate at the address location indicated by '*nnnn*'. Check & replace batteries.
 BRN ERR:nnnn	A burn error occurred at the address indicated by '*nnnn*'. This is often caused by specifying a blank or non-ZEPROM destination page.

DECHEX Convert decimal value in Reg X into hexadecimal

XROM 09,12

Purpose Converts the decimal address or word value stored in Reg X into hexadecimal format. The function takes the absolute integer portion of the number, and checks that it is a valid HP-41 ROM address or word, then converts it into the equivalent hexadecimal alpha string. The original value is saved to LASTX.

Input The user must supply a value for the following argument:

Reg X:	*decimal-number* (*nnnnn*)	Only the absolute integer portion of the number is used. DECHEX will convert numbers from 0 to 65535 ('0h' to 'FFFFh').

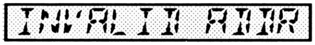
Output **For a valid decimal number:**

Reg X:	*hexadecimal-string* (*hhhh*)	The equivalent hexadecimal alpha string '0h' to 'FFFFh'
Reg L:	*decimal-number* (*nnnnn*)	The original decimal number in Reg X is saved to LASTX.

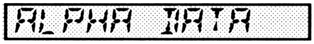
For an invalid decimal number: (Registers are not changed)

Reg X:	*decimal-number* (*nnnnn*)	The original number *nnnnn*.

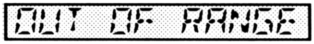
Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.


INVALID ADDR

A negative or other invalid address value was specified in Reg X.

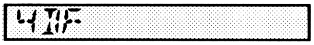

ALPHA DATA

The value in Reg X was already an Alpha value.


OUT OF RANGE

The value in Reg X to be converted exceeds 65535 ('FFFFh').

Example This example shows the key sequence needed to convert a decimal number of 1247 into a hexadecimal number.

Key Sequences	Description
[1] [2] [4] [7]	Input the decimal number into the X register.
[XEQ] [ALPHA] [D] [E] [C] [H] [E] [X] [ALPHA]	Execute the DEC to HEX function to convert the value.
 4DF	Display shows the hexadecimal equivalent, which has been stored into Reg X as an alpha string.

ENABLEP — Enable primary bank of ROM
XROM 09,13 — whose page is in Reg X

Purpose ENABLEP takes a ROM page number from Reg X and swaps the banks in that bank-switched ROM page such that the primary bank (bank 1) is enabled and the secondary bank (bank 2) is disabled. This function, together with ENABLES, permits the user to swap between the two banks of any bank-switched ROM plugged into the HP-41.
This program is provided specifically for use whilst programming ZEPROM modules. With the aid of this function, users can control the bank-switching of any module currently plugged into the HP-41. This may prove useful with ZENROM's MCED function for examining and debugging code that you have already burnt into switched banks.

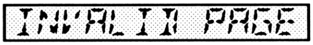
Cautions The ENABLEP instruction only acts upon the particular ROM module whose page number has been specified in Reg X. However, because of the way the module operates, both primary pages in that module are enabled at the same time.

Input The user must supply a value for the following argument:

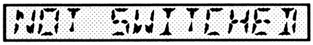
Reg X:	*target-page*	Target page specified as a floating point value, or a hexadecimal string.

Output None.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.


INVALID PAGE

An invalid page was specified.


NOT SWITCHED

The page in the ROM specified in Reg X is not bank switching.

ENABLES

Enable secondary bank of ROM whose page is in Reg X

XROM 09,14

Purpose ENABLES takes a ROM page number from Reg X and swaps the banks in that bank-switched ROM page such that the secondary bank (bank 2) is enabled and the primary bank (bank 1) is disabled. This function, together with ENABLEP, permits the user to swap between the two banks of any bank-switched ROM plugged into the HP-41.

This program is provided specifically for use whilst programming ZEPROM modules. With the aid of this function, users can control the bank-switching of any module currently plugged into the HP-41. This may prove useful with ZENROM's MCED function for examining and debugging code that you have already burnt into switched banks.

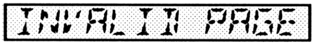
Cautions The ENABLES instruction only acts upon the particular ROM module whose page number has been specified in Reg X. However, because of the way the module operates, both secondary pages in that module are enabled at the same time.

Input The user must supply a value for the following argument:

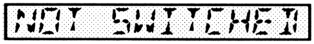
Reg X:	*target-page*	Target page specified as a floating point value, or a hexadecimal string.

Output None.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.


INVALID PAGE

An invalid page was specified.


NOT SWITCHED

The page in the ROM specified in Reg X is not bank switching.

FRSPC? Determine address and amount of free space
XROM 09,15 of page in Reg X

Purpose Calculate the free (unused) space remaining between the FAT and the beginning of the bank-switching code/normal interrupts. If the specified page contains free space, its starting address is placed into Reg Y, and the number of free bytes and remaining FAT entries is placed into Reg X.

Cautions FRSPC? cannot check whether or not the specified page contains a plug-in module, or if that module is a ZEPROM.

Input The user must supply a value for the following argument:

Reg X:	*target-page*	Target page specified as a floating point value, or hexadecimal string.

Output Output depends upon whether it was executed from the keyboard, or under program control. See page 39 for full details of comparison tests.

Under Keyboard Execution:	Displays a YES or NO answer.
Has free space (TRUE)	Displays 'YES' answer. **Reg Y:** free space starting address in floating point or hexadecimal. **Reg X:** free bytes (*nnnn*) and remaining FAT entries (*fff*) in format of a floating point value '*nnnn* . *fff*' **Reg L:** *target page* saved from Reg X
No free space (FALSE)	Displays 'NO' answer . **Reg X:** remains unchanged (*page*).
Under Program Control:	Doesn't display an answer, rather executes or skips the next program instruction depending upon the result. Follows 'DO IF TRUE' rule:
Has free space (TRUE)	Perform the next program step. **Reg Y:** free space starting address in floating point or hexadecimal. **Reg X:** free bytes (*nnnn*) and remaining FAT entries (*fff*) in format of a floating point value '*nnnn* . *fff*' **Reg L:** *target page* saved from Reg X
No free space (FALSE)	Skips the next program step. **Reg X:** unchanged (*target page*).

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

INVALID PAGE

An invalid page was specified.

HEXDEC — Convert hexadecimal string into decimal number

XROM 09,16

Purpose Converts a hexadecimal alpha string stored in Reg X into its floating point decimal integer equivalent. The function accepts an alpha string of up to 4 characters, checks that it is a valid HP-41 ROM address or word, then converts it. The original value is saved to the LASTX register.

Input The user must supply a value for the following argument:

Reg X:	*hexadecimal-string* (*hhhh*)	A hexadecimal string of maximum 4 characters of value '0h' to 'FFFFh'.

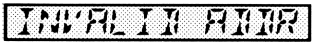
Output **For a valid hexadecimal string:**

Reg X:	*decimal-number* (*nnnnn*)	The equivalent decimal integer value 0 to 65535.
Reg L:	*hexadecimal-string* (*hhhh*)	The original hexadecimal alpha string in Reg X is saved to LASTX.

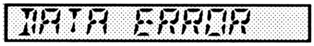
For an invalid hexadecimal string: (Registers are not changed)

Reg X:	*hexadecimal-string* (*hhhh*)	The original hexadecimal string.

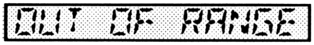
Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.


INVALID ADDR

A negative or other invalid address value was specified in Reg X.


DATA ERROR

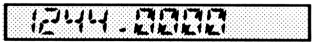
Reg X contains a decimal value. Store as an alpha string.


OUT OF RANGE

The value in Reg X to be converted exceeds 'FFFFh' (65535).

Example This example shows the key sequences needed to convert a hexadecimal value of 4DCh into a decimal number.

Key Sequences	Description
ALPHA ■ 4 D C ■ ASTO . X ALPHA	Input the hexadecimal number into the alpha register. Then store that into Reg X as an alpha string.
XEQ ALPHA H E X D E C ALPHA	Execute the HEX to DEC function to convert the value.


1244.0000

Display shows the number converted into decimal.

ILBURN — Burn page using ROM image from HP-IL mass storage file

XROM 09,17

Purpose Retrieves the named file containing an entire 4K ZEPROM page image from HP-IL mass storage medium and burns that image direct into the specified ZEPROM page. The function will work with any HP-41 LIF (Logical Interchange Format) medium such as cassette or 3½" micro-disc.

Theory ILBURN expects that specified mass storage file contains an HP-41 ROM image in the Eramco compressed 640 register/20 record format. This is the format that has been adopted as a *de facto* standard by most Machine Language Development Laboratory (MLDL) operating systems, e.g. the SAVEROM function in the Eramco MLDL operating system. This format has been chosen because it provides both maximum compression and compatibility with already established products.

Cautions ILBURN requires that the HP-41 has been correctly set up ready for burning of a ZEPROM module, with a ZEPROM Voltage Converter and a ZEPROM connected and the ZVC's programming switch set to the ON position. It also requires that an HP-IL module and mass storage drive are connected and turned-on.

ILBURN will only search the directory of the currently SELECTed HP-IL drive. If more than one exists, the user should SELECT the intended drive before executing the function.

Input The user must supply values for the following arguments:

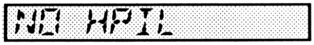
Reg X:	*target-page*	Target page specified as a floating point value, or a hexadecimal string.
Alpha:	*mass-storage-file-name*	Valid mass storage file name of maximum 7 characters.

Output Stack registers are unchanged unless an error occurs during burning. See page 40 for details of outputs during burning. In the event of a burn failure the stack registers are modified as follows.

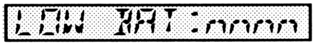
Burn failed:	**Alpha:** unchanged *(file name)* **Reg X:** burn failure address in hexadecimal format. **Reg L:** Target pagebeing burnt.

Errors Refer to the *HP-41 & HP-IL Owner's Manuals* for details of standard HP-41 and HP-IL messages.

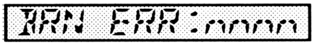
INVALID PAGE	An invalid page was specified.
NONEXISTENT	A non existent file name was specified


NO HPIL

No HP-IL module is connected to the HP-41.


LOW BAT:nnnn

A low battery has caused burning to terminate at the address location indicated by 'nnnn'. Check & replace batteries.


BRN ERR:nnnn

A burn error occurred at address indicated by '*nnnn*'. This is often caused by specifying a blank or non-ZEPROM destination page.

Example This example uses **ILBURN** to burn 4 ZEPROM pages with images stored as files on the currently SELECTed HP-IL mass storage device. It assumes that the ZVC and ZEPROM module are plugged-in with the programming switch already set to ON and that the HP-IL and mass storage device are connected and turned-on. The mass storage files to be loaded are called: *file*0, *file*1, *file*2, *file*3. It also expects the first page number (in decimal format) to be in Reg X and takes the base part of the file name from Alpha and appends '0,1,2,3' in turn..

```
01♦LBL "BURN16K       Expects Reg X: base page number
                          Alpha: "file" (basis of filename)
02 FIX 0              Ensure ARCL is only integer part
03 CF 29                without a decimal point.
04 .003               Load control number for ISG loop
05 ASTO Z             Store filename in Reg Z.
06♦LBL 00
07 INT                compute next page number
08 CLA                Clear Alpha for filename building
09 ARCL Z             Recall base filename to alpha
10 ARCL X             Append next page number to alpha
11 X<>Y               Swap next page & base page
12 RCL L              Recall counter
13 RDN                Reg X=base page, Reg T= counter
14 +                  Add base to next page number
15 ILBURN             Burn the image into ZEPROM bank
16 R↑                 Base page to Y & counter to X
17 ISG X              Increment loop control number
18 GTO 00             Loop back to get other pages
19 RTN                Done all four pages
```

ILSAVE

Save ZEPROM page image into HP-IL mass storage file

XROM 09,18

Purpose Saves the entire 4K ZEPROM page specified in Reg X into an HP-IL mass storage medium file. The function will work with any HP-41 LIF (Logical Interchange Format) medium such as cassette or 3½" micro-disc.

Theory ILSAVE saves the specified page into a mass storage file in the Eramco compressed 640 register/20 record format. This is the format that was adopted as a *de facto* standard by most Machine Language Development Laboratory (MLDL) operating systems, e.g. the GETROM function in the Eramco MLDL operating system. This format has been chosen because it provides both maximum compression and compatibility with already established products.

Cautions The data is written to the currently selected mass storage device present on HP-IL, and if more than one device exists the intended device can be specified by the SELECT function in the HP-IL module. ILSAVE expects that the HP-IL mass storage medium (micro-disc, cassette, etc.) has already been initialised by the NEWM instruction, has a free catalogue entry and enough free space for a file of 5120-bytes (640 registers).

Input The user must supply values for the following arguments:

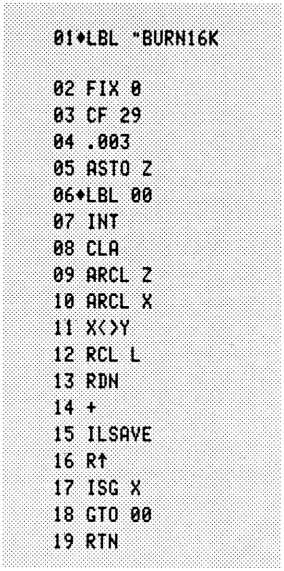
Reg X:	*target-page*	Target page specified as a floating point value, or a hexadecimal string.
Alpha:	*mass-storage-file-name*	Valid mass storage file name of maximum 7 characters.

Output Registers remain unchanged.

Errors Refer to the *HP-41 & HP-IL Owner's Manuals* for details of standard HP-41 and HP-IL messages.

INVALID PAGE	An invalid page was specified.
NO HPIL	No HP-IL module is connected to the HP-41.

Example This example uses ILSAVE to save an entire straight 16K ZEPROM into four 4K image files on the currently SELECTed HP-IL mass storage device. It assumes that ZEPROM module is plugged-in and that the HP-IL and mass storage device are connected and turned-on. The mass storage files will be saved as: FILE0, FILE1, FILE2, FILE3. It expects the first page number (decimal format) to be in Reg X and takes the base part of the file name from Alpha.



```
01♦LBL "BURN16K        Expects Reg X: base page number
                          Alpha: "file" (basis of filename)
02 FIX 0               Ensure ARCL is only integer part
03 CF 29                 without a decimal point.
04 .003                Load control number for ISG loop
05 ASTO Z              Store filename in Reg Z.
06♦LBL 00
07 INT                 compute next page number
08 CLA                 Clear Alpha for filename building
09 ARCL Z              Recall base filename to alpha
10 ARCL X              Append next page number to alpha
11 X<>Y                Swap next page & base page
12 RCL L               Recall counter
13 RDN                 Reg X=base page, Reg T= counter
14 +                   Add base to next page number
15 ILSAVE              Save the image onto mass storage
16 R↑                  Base page to Y & counter to X
17 ISG X               Increment loop control number
18 GTO 00              Loop back to get other pages.
19 RTN                 Done all four pages
```

INITPG — Initialise a ZEPROM Page

XROM 09,21

Purpose Initialises (formats) a ZEPROM page specified by the value in Reg Y. Arguments for bank switching, the XROM number and maximum number of Function Address Table (FAT) entries are taken from Reg X, while the ROM-header and trailer information are taken from Alpha.

If a negative XROM-id is specified, INITPG will set up the page by loading the necessary switching code at location *x*FC7h through *x*FCAh and set the switched status bit at location *x*FFDh.

Action Before initialisig the page INITPG checks the page location *x*000h to determine whether or not the specified page is blank. If location 0000h is non-zero, then INITPG terminates.

Cautions The argument for FAT entries should be the maximum number of functions and programs that you will be putting into that page plus one for the header, if desired.

Because the HP-41 cannot perform a CAT 2 on a page with blank FAT entries, as the FAT chain and links are incomplete, **do not** perform a CAT 2 on that page until the FAT is completely filled with functions/programs. If a CAT 2 is performed with blank entries, the HP-41 may crash.

If any ZEPROM page is initialised as being bank-switched, then **all** banks of all pages in that ZEPROM **must** be bank-switching. This applies whichever bank-switched configuration is being used. After initialising the first page in a ZEPROM, use ADDBSW to just add the switching code into all other banks until you are ready to INITPG those banks with XROM numbers, headers, FATs, etc..

Input The user must supply values for the following arguments:

Reg Y:	*target-page*	Target page specified as a floating point value, or a hexadecimal string.
Reg X:	*XROM .functions*	A floating point number in the format *'nn.fff'* where: *'nn'* is an XROM-id number from 01 to 31. A negative XROM-id indicates that the page is to be bank-switched. *'fff'* is a value between 0 & 64 indicating the maximum FAT entries desired. If '0' is specified, INITPG defaults to 64 entries.

© Zengrange Ltd - 1988

Alpha:	*header* , *trailer*	Two alpha strings separated by a comma. The *header* can be up to 11 characters but the *trailer* **must** be 4 characters. Valid characters are those of byte values of 20h to 5Fh. True page headers should be of 8 or more characters if they are to appear in a 41CX CAT 2. Normally have a '-' as their first character.

Output Stack registers are unchanged unless an error occurs during burning - See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owner's Manuals* for details of standard messages.

INVALID PAGE

Page specified in Reg Y is not blank at location x000h. If it contains data, the page is considered invalid.

LOW BAT:nnnn

A low battery has caused burning to terminate at the address location indicated by '*nnnn*'. Check & replace batteries.

BRN ERR:nnnn

A burn error occurred at the address indicated by '*nnnn*'. This is often caused by specifying a blank or non-ZEPROM destination page.

DATA ERROR

Reg X contains an invalid value for either XROM-id or number of functions for the FAT.

Example This example illustrates the key sequences used to initialise page 8 of ZEPROM with an XROM-id of 31 and a FAT of 25 entries. In addition, a header of -MYROM 1A and trailer of MR1A are burnt into the page.

Key Sequences	Description
[ALPHA] **-MYROM1A,MR1A** [ALPHA]	Input the *page-header* , *trailer* into the alpha register.
[8] [ENTER]	Input the *target-page number* and duplicate it into Reg Y.
[3] [1] [.] [0] [2] [5]	Input the *XROM number* , *number of functions* into Reg X.
[XEQ] [ALPHA] **INITPG** [ALPHA]	Execute the [INITPG] function

PB09 — Enable primary bank of Programmer ROM module

XROM 09,22

Purpose Disables the secondary bank (bank 2) and enables the primary bank (bank 1) of the PROGRAMMER ROM module. This function, together with SB09, enables users to switch between the banks of the PROGRAMMER ROM if this module has been made bank switching by the user.

Theory Because HP-41 bank-switching has been implemented such that only the particular module containing the bank-switching instruction will obey it, each bank of each page in a bank-switching must contain the bank-switching functions. As a user may have more than one ZEPROM module plugged in at any time, and each of these functions must have a unique XROM-id to avoid confusion between them, the functions all adopt the XROM-id of the page into which they are loaded. The general form of these functions is PBxx and SBxx, *see separate entries*. When loaded into an initialised page by ADDMCF the functions take on the XROM-id of that page. Therefore, in the PROGRAMMER ROM (XROM-id of 09), these functions appear as PB09 and SB09.
These functions have been provided in the PROGRAMMER ROM in case a user decides to make that module bank-switching, and thus allow the user to switch between the primary and secondary banks.
See additional explanation about bank-switching functions on page 39.

Input None.

Output None.

Errors None.

PBxx

Enable primary bank of ROM with XROM-id of xx

Purpose Disables the secondary bank (bank 2) and enables the primary bank (bank 1) of the bank-switched ROM page whose XROM-id number is specified in Reg X. This function, together with SBxx, enables users to switch between the two banks of any particular bank-switched ROM as desired.

Theory Because HP-41 bank-switching has been implemented such that only the particular module containing the bank-switching instruction will obey it, each bank of each page in a bank-switching must contain the bank-switching functions. As a user may have more than one ZEPROM module plugged in at any time, and each of these functions must have a unique XROM-id to avoid confusion between them, the functions all adopt the XROM-id of the page into which they are loaded. When loaded into an initialised page by ADDMCF the functions take on the XROM-id of that page. Therefore, in a ROM page of XROM-id of 28), these functions appear as PB28 and SB28.
See additional explanation about bank-switching functions on page 39.

Input None.

Output None.

Errors None.

PGX=Y? Compare two 4K ROM pages for equality

XROM 09,23

Purpose Performs a comparison of the ROM page specified by Reg X against the master specified in Reg Y. If both images are not identical, then the location address of the first mismatched code is returned to Reg X.

Input The user must supply values for the following arguments:

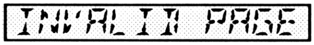
Reg Y:	*First-page*	Master 4K-page specified as floating point value, or hexadecimal string.
Reg X:	*Second-page*	4K-page to be verified specified as either a floating point value, or hexadecimal string.

Output Varies according to whether the function was executed from the keyboard, or under program control. See Page 39 for details of the general format of comparison tests.

Under Keyboard Execution:	Displays a YES or NO answer.
Pages identical (TRUE)	Displays 'YES' answer. **Reg X** & **Y** remain unchanged.
Pages not identical (FALSE)	Displays 'NO' answer **Reg Y:** unchanged (*1st-page*). **Reg X:** address of first mismatched code. Address is either floating point or hexadecimal string depending upon original value in X. **Reg L:** *Verified page*. Original page number moved from Reg X.
Under Program Control:	Doesn't display an answer, rather executes or skips the next program instruction depending upon the result. Follows 'DO IF TRUE' rule.
Pages identical (TRUE)	Performs the next program step. **Reg X** & **Y** remain unchanged.
Pages not identical (FALSE)	Skips the next program step. **Reg Y:** unchanged (*Master-page*). **Reg X:** address of first mismatched code. Address is either floating point or hexadecimal string depending upon original value in X. **Reg L:** *Verified page*. Original page number moved from Reg X.

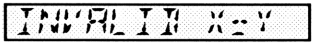
Errors

Refer to the *HP-41 Owner's Manuals* for details of standard messages.



INVALID PAGE

An invalid page was specified.



INVALID X=Y

The origin and destination are identical.

Example

This example compares a 16K straight (non-bank-switched) ZEPROM in port 1 or 2 with another straight 16K ZEPROM module in port 3 or 4.

`01♦LBL "CMPZEP`	Compare two ZEPROMs
`02 12.1`	Load port 3 & 4 control number (page 12 to 15). Test set so that 'skip' will never happen
`03 8.011`	Load port 1 & 2 control number (pages 8 to 11)
`04♦LBL 00`	Start comparison loop
`05 PGX=Y?`	Test equivalence of pages
`06 GTO 01`	These two pages are equal
`07 "PAGE "`	Pages are not equal. Load start of error message
`08 DECHEX`	Convert page number to hex for appending to alpha string
`09 ARCL X`	Append first page number to alpha
`10 "⊢≠`	Append the 'not equals' sign
`11 X<>Y`	Get other page number to Reg X
`12 DECHEX`	Convert to hexadecimal
`13 ARCL Y`	Append it to alpha
`14 GTO 02`	Go to message view routine
`15♦LBL 01`	Pages tested equal so far
`16 ISG Y`	Increment page number of second ZEPROM module
`17 ISG X`	Increment page number of first ZEPROM module. Skip next step if done.
`18 GTO 00`	Not finished yet, so loop back
`19 "ZEPROMS ARE =`	All done. All pages are the same
`20♦LBL 02`	Display message routine
`21 AVIEW`	Display message in alpha
`22 RTN`	

PRGMLN — Return program length and number of LBLs

XROM 09,24

Purpose Calculates the length of the user code program specified in the Alpha register and returns the number of bytes and FAT entries needed for that program to be stored in a ZEPROM page. This information can subsequently be compared with the output of **FRSPC?** to determine whether or not adequate space and free FAT entries remain for that program to be loaded into.

The Alpha register may contain the name of any global alpha label in the intended program, or if blank **PRGMLN** takes the current program.

Cautions **PRGMLN** calculates the length of the specified program, irrespective of whether or not that program is currently packed. To determine the length of a program that will be loaded with **BURNUC**, the program should be PACKed before executing **PRGMLN**.

Input The user must supply values for the following arguments:

Alpha:	*program-name*	An alpha string containing the program's name or any global alpha LBL within that program. If blank the current program is to be used).

Output Register X is modified as follows:

Reg X:	*bbbbb . ccc*	A floating point number in the format *'bbbbb. ccc'* where: *'bbbbb'* is the number of bytes in a ROM page required to insert the program, and *'ccc'* is the number of FAT entries required for all global alpha LBLs in the program.

Errors Refer to the *HP-41 Owners Manuals* for details of standard messages.

NONEXISTENT	A non-existent global alpha LBL was specified.
M-CODE LBL	The program LBL in Alpha is that of a machine code function in ROM or in the HP-41 operating system.
LBL TOO LONG	The program LBL in Alpha exceeds 7 characters.

Example This example uses PRGMLN to compute the size and FAT entries of a program named in Alpha, then uses FRSPC? to check whether or not enough free space and FAT entries remain in the ROM page specified in Reg X.

```
01♦LBL "LENSPC"       Compute length/FAT entries for
                         program in Alpha and check
                         available space/FAT in ROM
                         page specified in Reg X.
02 PRGMLN             Get program length & FAT entries
                         (RegX = bytes.FAT entries)
03 INT                Take number of bytes (integer)
04 LASTX              Recall the value
05 FRC                Take number of FAT entries needed
06 RCL Z              Recall the page number
07 FRSPC?             Room left ? (Reg X = bytes.FAT;
                         Reg Y = start address)
08 X<>Y               Swap start address to Reg X
09 RDN                Roll down (Reg X = bytes.freeFAT)
10 FRC                Number of free FAT entries
11 LASTX              Recall last value
12 INT                Number of free bytes remaining
13 R↑                 Reg Y=free space; Reg X=prog length
14 "NO ROOM"          Load "no room" message
15 X>Y?               Program length > space ?
16 GTO 01             Yes, so go tell him "no room"
17 RDN                Roll stack down
18 RDN                Reg Y = labels, Reg X = freeFAT
19 "FAT OVERFLOW"     Load "overflow" message
20 X<Y?               LBLs required less than freeFAT?
21 "SPACE IN PG"      Yes, so load "space" message
22 ♦LBL 01
23 AVIEW              Go show him the result
24 RTN                Terminate
```

READWD

Read word from ROM page in Reg X

XROM 09,25

Purpose Reads (recalls) the content of the word at any HP-41 ROM page address (specified in Reg X) and returns that word value to Reg X.

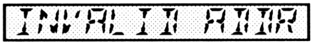
Input The user must supply a value for the target address in floating point decimal or hexadecimal format. Note that READWD accepts all valid HP-41 addresses from 0000h to FFFFh (0 to 65535) and therefore can be used to read words in HP-41 system addresses in addition to the four I/O ports. E.g: using 73F2h (29682) causes read of word at address 3F2h in the HP-IL/Mass Storage ROM, if the HP-IL module is plugged into HP-41.

Reg X: *target-address*	16-bit target address specified in floating point, or hexadecimal.

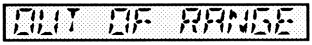
Output The data word read by READWD is returned in the same format as that used for the original input of the target address and therefore can be floating point decimal or hexadecimal.

Valid Address:	**Reg Y:** *target-address* from Reg X. **Reg X:** *Word* read from address.
Invalid Address:	**Reg X:** *target-address* as input.

Errors Refer to the *HP-41 Owners Manuals* for details of standard messages.


INVALID ADDR

A negative or other invalid address value was specified in Reg X. Check that positive and/or valid positive characters were used.


OUT OF RANGE

The value in Reg X to be converted exceeds 'FFFFh' (65535).

Example Uses READWD in verifying the checksum currently burnt in the ROM page specified in Reg X against that calculated for the page by CHKSUM.

```
01♦LBL "VERSUM"     Program: Verify page checksum
02 CHKSUM           Get checksum & address
03 X<>Y             Move address to Reg X
04 READWD           Read that address in the ROM
05 X<>Y             Move actual checksum back to X
06 RDN              X=actual, Y=correct checksum
07 "GOOD "          Load default "good" message
08 X≠Y?             Is actual checksum NOT ok?
09 "BAD "           Yes, so replace with "bad" message
10 "⊢CHKSUM         Append "checksum" to message
11 AVIEW            Show message to user
12 RTN              End of routine
```

REBURN — Recover from error during burning of user code program

XROM 09,26

Purpose

Allows the user to overcome burn errors occurring whilst burning a user code program into a ROM page with BURNUC.

During burning with BURNUC, the address at which the function began burning the program is stored into Reg X. If a burn error occurs, the burn failure address is returned to Reg X and the start address of the program is moved to Reg L. It is essential that a user **does not** just re-try using BURNUC, since this will insert a new, duplicate image after that of the aborted attempt. To prevent this, use the REBURN instruction. To use REBURN, use LASTX to recall to Reg X the address of the first line of code burnt during the aborted attempt . The program name should still be in Alpha.

REBURN checks the program data already burnt into the ROM page and then continues burning the remainder of the program. However, it does not perform any checks for a packed and compiled program, so it is essential that the program is not edited before REBURN is used. A "synthetics" programmer may therefore also use REBURN to load an uncompiled and/or unpacked program into ZEPROM.

Cautions

This function is really intended only for use in recovering from BURNUC errors and **cannot** be used to recover from errors occurring during other burn operations. In most instances, other aborted functions can just be restarted.

Input

The user must supply values for the following arguments:

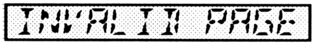
Reg X:	*start-address*	Address at which program started burning during aborted attempt with BURNUC. Either floating point value, or a hexadecimal string.
Alpha:	*program-name*	Name of user code program being burnt.

Output

Stack registers are unchanged unless an error occurs during burning - See page 40 for details of standard outputs following burn errors.

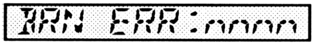
Errors

Refer to the *HP-41 Owner's Manuals* for details of standard messages.



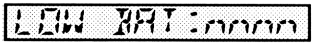
INVALID PAGE

An invalid page was specified.



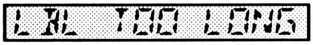
BRN ERR:nnnn

A burn error occurred at address indicated by '*nnnn*'. Commonly caused by specifying an empty or non-ZEPROM destination page.

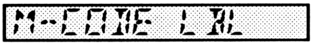


LOW BAT:nnnn

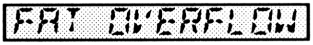
Low battery terminated burning at the address 'nnnn'. Check, replace the batteries and then REBURN.


LBL TOO LONG

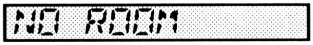
The program LBL in Alpha exceeds 7 characters.


M-CODE LBL

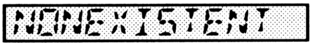
The program LBL in Alpha is that of a machine code function in ROM or in the HP-41 operating system.


FAT OVERFLOW

Not enough spare FAT entries for the number of global alpha LBLs in specified program. Change some to local LBLs.


NO ROOM

Not enough free space is left to load the program specified in Reg X.


NONEXISTENT

A non-existent global alpha LBL was specified in Alpha.

RRBURN — Burn ZENROM ROMREG+ formatted data from RAM into ZEPROM page

XROM 09,27

Purpose Burns data that has been copied from a Q-ROM device into main memory registers by the ZENROM module's Machine Code Editor (MCED).
RRBURN takes data from main memory registers according to the format of data input by the user. This allows the user to specify registers by their number, e.g. Reg:050, or by referencing an absolute memory address in the HP-41's main and extended memory.
This function has been provided primarily for ZENROM users. For more information on the ROMREG+ format, see the *ZENROM Owner's Manual.*

Theory In developing M-code programs or functions using a ZENROM module and a Q-ROM, MCED can be used to save (SVE) or restore (GET) M-code data in a compressed format in normal HP-41 memory registers. This data format, known as 'ROMREG+' format, is a superset of the ROMREG format used by many other M-code development packages and is based around storing five 10-bit ROM-words into each data register.

Caution Although the older ROMREG format only stored multiples of five ROM-words into each data register, this restriction does not apply to the ROMREG+ format used by ZENROM. ROMREG+ format therefore avoids the difficulties encountered where the last register of ROMREG data contains unwanted words. Because of the ROMREG format, these unwanted words are also transferred into ZEPROM during burning with RRBURN.
Extended memory registers can only be accessed in the absolute memory addressing mode. In register mode, a starting address input in floating point decimal, access is restricted to main memory registers.

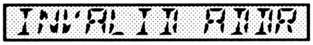
Input The user must supply values for the following arguments. The format of the starting address input into Reg Z will determine whether absolute (hexadecimal input) or register (floating point decimal) addressing is used.

Absolute Memory Addressing:		Hexadecimal input into Reg Z
Reg Z:	*1st-register-address*	Address of the first register to be burnt specified as a hexadecimal string.
Reg Y:	*Number-of-registers*	Quantity of registers specified as a floating point decimal number.
Reg X:	*target-address*	ZEPROM module target address specified as a floating point decimal number or hexadecimal string.
Data Register Addressing:		Floating point input in Reg Z.

Reg Z:	*1st-register-number*	Number of the first register to be burnt specified as a floating point decimal value. E.g. Reg 017
Reg Y:	*Number-of-registers*	Number of registers specified as a floating point decimal number.
Reg X:	*target-address*	ZEPROM module target address specified as a floating point decimal number or hexadecimal string.

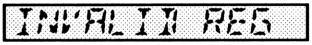
Output Stack registers are unchanged unless an error occurs during burning - See page 40 for details of standard outputs following burn errors.

Errors Refer to the *HP-41 Owners Manuals* for details of standard messages.



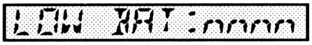
INVALID ADDR

Specified address is negative or otherwise invalid. Check that valid hexadecimal characters were used.



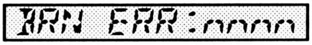
INVALID REG

Specified register is outside the current SIZE setting.



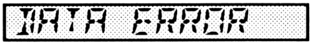
LOW BAT:nnnn

A low battery has caused burning to terminate at the address location indicated by '*nnnn*'. Check & replace batteries.



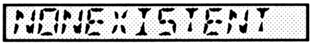
BRN ERR:nnnn

A burn error occurred at the address indicated by '*nnnn*'. This is often caused by specifying a blank or non-ZEPROM destination page.



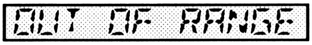
DATA ERROR

Specified data register contains data not in ROMREG+ or ROMREG format.



NONEXISTENT

A specified data register was not found during burning.



OUT OF RANGE

The decimal value input exceeds 65535.

SB09 — Enable secondary bank of Programmer ROM module

XROM 09,28

Purpose Disables the primary bank (bank 1) and enables the secondary bank (bank 2) of the PROGRAMMER ROM module. This function, together with PB09, enables users to switch between the banks of the PROGRAMMER ROM if this module has been made bank switching by the user.

Theory Because HP-41 bank-switching has been implemented such that only the particular module containing the bank-switching instruction will obey it, each bank of each page in a bank-switching must contain the bank-switching functions. As a user may have more than one ZEPROM module plugged in at any time, and each of these functions must have a unique XROM-id to avoid confusion between them, the functions all adopt the XROM-id of the page into which they are loaded. The general form of these functions is PBxx and SBxx, *see separate entries*. When loaded into an initialised page by ADDMCF the functions take on the XROM-id of that page. Therefore, in the PROGRAMMER ROM (XROM-id of 09), these functions appear as PB09 and SB09.
These functions have been provided in the PROGRAMMER ROM in case a user decides to make that module bank-switching, and thus allow the user to switch between the primary and secondary banks.
See additional explanation about bank-switching functions on page 39.

Input None.

Output None.

Errors None.

SBxx — Enable secondary bank of ROM with XROM-id of xx

Purpose Disables the primary bank (bank 1) and enables the secondary bank (bank 2) of the bank switched ROM page whose XROM-id number is specified in Reg X. This function, together with PBxx, enables users to switch between the two banks of any particular bank-switched ROM as desired.

Theory Because HP-41 bank-switching has been implemented such that only the particular module containing the bank-switching instruction will obey it, each bank of each page in a bank-switching must contain the bank-switching functions. As a user may have more than one ZEPROM module plugged in at any time, and each of these functions must have a unique XROM-id to avoid confusion between them, the functions all adopt the XROM-id of the page into which they are loaded. When loaded into an initialised page by ADDMCF the functions take on the XROM-id of that page. Therefore, in a ROM page of XROM-id of 28), these functions appear as PB28 and SB28.
See additional explanation about bank-switching functions on page 39.

Input None.

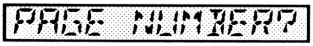
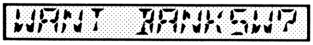
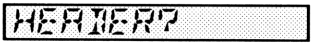
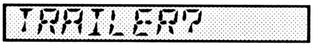
Output None.

Errors None.

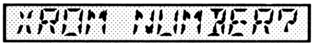
UCBURN — User Code Utility Program

XROM 09,29

Purpose Provides an automated, prompting routine for non-technical users to allow initialising and burning of user code program(s) into a ZEPROM page. Note that programs cannot be retrieved from extended memory or mass storage and that input of all values is in floating point decimal.

Step	Display	Key Sequences & Comments
1.		Ensure the ZVC and ZEPROM are properly connected and the ZVC is switched ON. In addition, ensure the HP-41 has a fresh set of alkaline batteries, and that the programs to be copied are in main memory. To start the ᵀUCBURN utility: [XEQ] [ALPHA] UCBURN [ALPHA]
2.	 PAGE NUMBER?	Input page number you want the programs loaded into. Remember ZEPROM is forced into 16K mode. 8 [R/S] If the page specified has already been initialised then goto step 10.
3.		Routine prompts for all initialisation parameters. If any other pages in ZEPROM are already initialised as bank-switched, then goto step 5. (In such cases new page automatically made bank-switching).
4.	 WANT BANKSW?	If module is to be bank-switching, input 1, otherwise input 0. 0 [R/S]
5.	 HEADER?	Header name (if required). Should be at least 8 characters long to show up in a CAT 2 on an HP-41CX. -MY ROM 1A [R/S]
6.	 TRAILER?	Page trailer (if required). Should be exactly 4 characters in length. MY1A [R/S]

7.

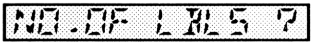


XROM NUMBER?

XROM number for page. Should be unique to your HP-41 configuration; i.e. not used by any other module.

21 [R/S]

8.

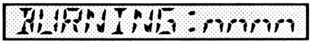


NO.OF LBLS ?

Add number of m-code functions to number of global alpha labels in your programs to give total number of labels. **Remember** that once initialised, this cannot be changed! If unsure of exact number, it is best to over-estimate! Don't add entries for the header or bank-switching functions that will be automatically loaded, since they are added by the program.

40 [R/S]

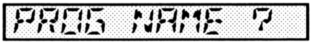
9.



BURNING:nnnn

Program initailising page with FAT, XROM-id (plus header & trailer if specified). Functions PBxx, SBxx, BGTOxx, BXEQxx & BRTNxx are also loaded if bank-switching.

10.

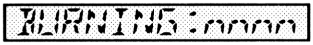


PROG NAME ?

ᵀUCBURN ready to load user code programs. Input name (global alpha label) of program to load. Specified program is compiled and burned.

PROG1 [R/S]

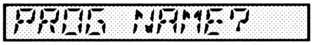
11.



BURNING:nnnn

The program is now being loaded into the ZEPROM.

12.



PROG NAME?

If more programs to burn, repeat from step 10 until either all your programs are burned or you need to copy in more from mass storage.

Once finished, respond to the PROG NAME? prompt by pressing [R/S] without any input. This will terminate the program.

[R/S]

Sub-routines:

ᵀUCBURN utility has 2 subroutines, ᵀINIT and ᵀINITP, used to initialise ZEPROM pages. In essence they cover step 2 through step 9 of the above example, except that ᵀINIT is intended for calling from the keyboard and ᵀINITP to be called from another program.

Register & Flag usage:

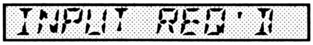
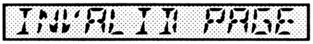
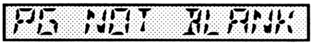
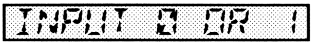
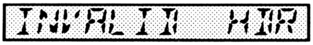
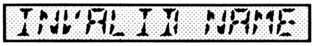
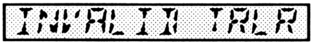
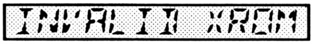
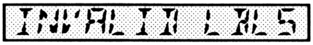
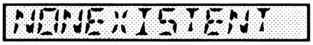
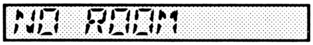
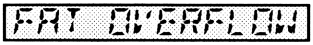
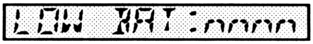
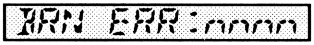
ᵀUCBURN and the ᵀINIT and ᵀINITP subroutines use 6 registers and two flags:

Register 00: Page number
Register 01: XROM number
Register 02: Number of labels
Register 03: Header (leading chars.)
Register 04: Header (trailing chars.)
Register 05: Trailer

Flag 19: set if INITP called, cleared if INIT

Flag 20: set if bank-switching, cleared if not

Error Messages:

 INPUT REQ'D	Input was required at the prompt. The program re-prompts for input.
 INVALID PAGE	Page number specified was invalid. The program re-prompts for correct input.
 PG NOT BLANK	ᵀINIT only. The page number specified is not blank. Program re-prompts for another page.
 INPUT 0 OR 1	Incorrect or no input at the WANT BANKSW? prompt. Program re-prompts for correct input.
 INVALID HDR	The page header entered was too long (max 11 chars). Program re-prompts for correct input.
 INVALID NAME	Program name was more than 7 characters long. Re-prompts for correct input.
 INVALID TRLR	The trailer entered was not exactly 4 characters long. Program re-prompts for correct input.
 INVALID XROM	The XROM ID entered was not in the range 1 to 31. The program re-prompts for correct input.
 INVALID LBLS	Either: number of labels input was 0, or number input added to header & bank-switch functions (if specified) is greater than 64. Re-prompts.
 NONEXISTENT	Program not found. Name either wrong, or not in main memory. Re-prompts for input. If necessary, copy in from mass storage & restart.
 NO ROOM	Page has not enough space to burn program. Abort, use new page, or load smaller programs.
 FAT OVERFLOW	Not enough free FAT entries left for program. Remove a few global labels or abort and restart specifying a new page. Re-prompts for input.
 LOW BAT:nnnn	Battery too low to continue burning. Replace batteries and use the REBURN to finish off burning this program. Then restart ᵀUCBURN.
 BRN ERR:nnnn	Error in burning program. Check for wrong page number input. Restart with correct page.

Appendix A

Technical Data

Programming ZEPROM

In common with other HP-41 add-on memory devices, such as Q-ROMs, the ZEPROM module uses the WMLDL (040h) opcode as its write instruction. However, it should be noted that programming a ZEPROM location correctly is not as simple as programming a location in a Q-ROM device. A Q-ROM is essentially a RAM based device, and as such, only requires a very short programming cycle - in fact just one write instruction will correctly program a RAM location. However, EPROM type devices are different in that they require both a long write cycle and a much higher than normal voltage for programming.

Programming Voltage

The necessary programming voltage (12.5V) is generally supplied by the programming fixture. In the case of the ZEPROM Voltage Converter this voltage is generated from the HP-41's own 6-volt batteries and supplied to the module when the ZVC's programming switch is set to the ON position. Whenever this programming voltage is applied to the ZEPROM module, the configuration bits programmed into the chip are ignored and the module is forced into a straight 16K mode. This ensures that all four cores are available while programming takes place.

Programming Cycle

Whenever the WMLDL instruction is issued on the 41 bus the ZEPROM module compares the address stored in C[6:3] (which is obtained from the DATA line) with its own address space. If the address to be programmed is within the ZEPROMs address space, then the ZEPROM starts its programming cycle. During a programming cycle the ZEPROM module goes off-line, (i.e. it does not respond to any instructions from the bus) and it sets the HP-41's peripheral flag 4 to indicate it has entered this state. The chip has to go off-line to ensure that the address loaded on the EPROMs address lines does not alter during the write cycle. The ZEPROM module stays off-line for a total of 16 HP-41 instruction cycles (typically 2.5ms). Once the cycle is completed, the ZEPROM comes back on-line and peripheral flag 4 is reset.

As with most EPROM devices there is a recommended 'Intelligent' or 'Quickburn' algorithm for programming the device; this algorithm requires that each location is programmed until the data read back from that location is the same as the data written. Then, in order to prevent 'bit drop-out' after the programming voltage has been removed, the location should be programmed twice more. Naturally, should the ZEPROM module fail to program correctly within a given number of tries then the programming operation should be aborted.

To illustrate the typical implementation of this algorithm, we have shown it in the following pseudo-code:

```
Reset retry counter
DO
            Load address and data
            Start write cycle (WMLDL)
                      DO
                      UNTIL  peripheral flag 4 clear
                      END
            Increment retry counter
            Read data from address
UNTIL
            data read = data written   OR   retry counter = 32
END

IF          data read = data written
THEN        burn successful
ELSE        burn failure
```

Programming Restrictions

Because the ZEPROM module is taken off-line during the programming cycle it would be impossible for the module to program itself since the instructions following a WMLDL would not be seen. Therefore, if the WMLDL instruction is issued from the same ZEPROM module as that being programmed, the ZEPROM will ignore the write instruction. The PROGRAMMER ROM circumvents this restriction by making an operating system call to initiate the write cycle.

HP-ROM Compatibility

The configuration of HP roms differs slightly from the configuration of a ZEPROM. An HP rom comprises just three 4K cores (total 12K) whereas the ZEPROM has four (total 16K).

With an HP ROM, each 4K core in the ROM has the following configuration options:

<table>
<tr><td colspan="3"><b>Enabled</b> or <b>Disabled</b></td></tr>
<tr><td><b>Hard addressed</b></td><td colspan="2"><b>Port addressed</b></td></tr>
<tr><td>Address</td><td>Lower page</td><td>Upper page</td></tr>
<tr><td colspan="3"><b>Permanent / Primary Bank / Secondary Bank</b></td></tr>
</table>

The options for HP ROMs are further explained as follows:

- If a core is *disabled*, it is not seen by the HP-41.
- If a core is *hard configured,* a configuration address must supplied.
- If the core is *port configured*, it must be configured for the upper or lower page in that port's address space.
- Each core, whether *hard* or *soft configured*, can also appear as:

  Permanent Bank Always present to the HP-41
  Primary Bank Present at power-up and enabled with ENBANK1; disabled with ENBANK2
  Secondary Bank Not present at power-up but enabled with ENBANK2; disabled with ENBANK1

Note, however, that the ENBANKx instruction only affects the module in which it is executed, but affects all cores in that ROM.

For ZEPROM, the situation is different. This is mainly because ZEPROM must emulate a number of different possibilities and must also be capable of being reconfigured under software control. HP's ROMs, being mask programmed ROMs rather than EPROMs, are hard configured and can never be changed.

- ZEPROM cannot have disabled cores; all cores being always enabled. However, it is possible to simulate disabling certain cores by configuring the zeprom as being bank-switched and then duplicating the data content of cores. *See chapter 2, page 8 for details.*

- ZEPROM cannot be hard addressed; its address always being dependant upon the port into which it is plugged. *See Chapter 1, page 1 & 2 for diagrams illustrating port addressing*. The position of each core within a port (i.e. in it's upper or lower page) is determined by the core number; cores 0 and 2 being the two lower pages and cores 1 and 3 being the two upper pages. This is not a restriction on the configuration of ZEPROM, but just means that the user must program the required data into the correct core.

- ZEPROM can emulate an HP ROM's permanent core by duplication of the primary bank code into the secondary bank. The selection of a bank being primary (Bank 1) or secondary (Bank 2) is controlled by the core number - cores 0 and 1 being primary banks, cores 2 and 3 being secondary banks.

In fact, the only configuration option directly offered by ZEPROM is a choice between a straight 16K configuration and a 16K bank-switched configuration. All other configurations, e.g. straight 8K, 12K bank-switched, are emulated by duplicating selected cores.

The following table illustrates how these various HP configurations can be emulated with ZEPROM:

HP-ROM Core Configurations	Emulation with ZEPROM
Enabled, Port addressed, Upper Page, Permanent	Core 1 duplicated into core 3
Enabled, Port addressed, Upper Page, Primary Bank (1)	Core 1
Enabled, Port addressed, Upper Page, Secondary Bank (2)	Core 3
Enabled, Port addressed, Lower Page, Permanent	Core 0 duplicated into core 2
Enabled, Port addressed, Lower Page, Primary Bank (1)	Core 0
Enabled, Port addressed, Lower Page, Secondary Bank (2)	Core 2

The ZEPROM hardware initially determines if the configuration is bank-switched or straight from the two most significant bits of data at address location *x*FFD in core 1. If both bits are clear, then the module is considered as straight 16K. If either or both bits are set, then the module is considered as being bank-switched. In a bank-switched configuration then either one or both of the two most significant bits of the data at address *x*FFD in core 3 should also be set. If this is not done, then ZEPROM will be able to select the secondary bank, but will not be able to return to the primary bank.

The use of the most significant bits of address *x*FFD complies with the bank-switching convention established by Hewlett-Packard; in which any bank switching page will have the most significant bit of the data at address *x*FFD in that page, set. Remember however that ZEPROM only references the bits in core 1 (and core 3) to determine if the entire ZEPROM is bank switching. The PROGRAMMER ROM software also uses an extension of the HP convention to determine if a page is bank-switched and also which page is currently selected:

Data Word at xFFD		Meaning of bits
bit-9	bit-8	
0	0	Page not bank-switched
0	1	Bank-switched, primary bank
1	0	Bank-switched, indeterminate bank (HP)
1	1	Bank-switched, secondary bank

The ZEPROM module is 100% code compatible with HP ROMs. This means that all user or M-code designed to run in an HP ROM will also run on ZEPROM. However, users should be aware of a slight incompatibility problem in the reverse direction. Because of a flaw in the chip currently being used by HP, all bank-switching instructions

(ENBANK1 and ENBANK2) must be preceded by an instruction in which the most significant bit is clear. The simplest way to do this is to precede the ENBANK1 or 2 with a `JNC +1` (003h) instruction. This restriction does not apply to the ZEPROM chip and code written specifically for ZEPROM can omit the `JNC +1`.

Erasing ZEPROM

The ZEPROM module can be erased by exposing the EPROM chip, via the window in the top of the module casing, to an intense ultra-violet (UV) light source. A dosage of 15W-seconds/cm^2 is required to completely erase the module. This dosage can be obtained by exposure to a UV lamp of wavelength 2537 Angstroms (Å) with an intensity of 12000 μW/m^2 for 15 to 20 minutes. During erasure, all filters should be removed from the UV source and the ZEPROM module should be positioned about 2.5cm from the source. Note, however, that EPROMs can also be erased by light sources having wavelengths shorter than 4000Å. Although erasure time will be considerably longer than with a UV source at 2537Å, nevertheless a prolonged exposure to fluorescent light or sunlight will eventually erase the chip, To realise maximum reliability, accidental erasure should be prevented by covering the module window with an opaque label.

!
Warning

Ultra violet light and UV-Erasers can be hazardous if not used strictly in accordance with the manufacturer's instructions. During erasure, all necessary precautions should be taken.

Electrical Information

The following current drain data has been established by direct measurements on an HP-41. Due to manufacturing tolerances in the HP-41, ZEPROM and ZVC, slight variances may be obtained in individual circumstances. All values quoted are typical.

ZEPROM Module	
Sleep state	3μA
Run mode	2.5mA

ZEPROM Voltage Converter	
Programming switch OFF	0
Programming switch ON	4.9mA

ZVC & ZEPROM - Programming	
Average	14.6mA
Peaks	15.1mA

(Note: the HP-41 consumes an additional 10 to 15mA in run mode)

Copyright & using ZEPROM

All copy, design, patent and ownership rights of this manual, its associated products, software, source code and descriptions shall remain with Zengrange Ltd and others. The provision of the software, and inclusion of any source code in this manual does not, and shall not, constitute its passing into the public domain. The User is permitted to use the software for purposes of burning ZEPROM modules, but neither the software, listings nor descriptions may be further distributed, modified or included in any other commercial products without the prior written approval and agreement of Zengrange.

!

Caution

The PROGRAMMER ROM contains functions that facilitate the copying of ROM software. However, since software is covered by copy, design or patent rights, it is illegal to make copies for distribution, duplication or modification unless permission has been granted by the holder of those rights. Obtaining such permission is the specific responsibility of the user.

In all cases where a Zengrange product is used to infringe copy, design or patent rights, the user shall bear the entire responsibility and assume all liability for infringement. Neither Zengrange Ltd, nor its agents, shall be in any way liable for the user's actions.

Appendix B

Annotated Listings

Utility Routines (RPN)

ᵀUCBURN provides for initialising and subsequently loading user code programs into a ZEPROM page. *See Chapter 5 for explanation.* The program also includes synthetic instructions (STO M, etc) and synthetic text, which is shown in hexadecimal format.

```
01♦LBL "UCBURN"         Entry point for loading user code programs
02 XROM "INITP"         Go initialise page if necessary

03♦LBL 12               Get name of program to burn
04 "PROG NAME"          Load "prog name" message
05 XEQ 07               Go get Alpha input from user
06 FC? 23               Did we get any input?
07 RTN                  No, so terminate the program
08 .                    Load zero into Reg X
09 RCL M                Was there any Alpha input (prog name) ?
10 X=Y?
11 RTN                  No, so terminate program
12 X<>Y                 Get zero into Reg X
13 RCL N                Did the user goof and input more than
14 X=Y?                   7 characters for the program name ?
15 GTO 13               Length okay, so go process it

16♦LBL 14               Error in program name returned
17 XEQ 15               Go get "invalid" message
18 "⊢NAME"              Append "name"
19 XEQ 10               Go show the user
20 GTO 12               Go and retry to get a valid program name

21♦LBL 13               Got the program name
22 SF 25                Trap any PRGMLN errors
23 PRGMLN               Get program length & FAT entries (RegX = bytes.FAT entries)
24 FS? 25               Did PRGMLN cause an error ?
25 GTO 13               No, so program length is in Reg X
26 "NONEXISTENT"        Yes, program not found, or M-code function
27 GTO 14               Go tell user about error

28♦LBL 13               Program length in Reg X
29 INT                  Take number of bytes (integer)
30 LASTX                Recall the value
31 FRC                  Take number of FAT entries needed (fraction)
32 RCL 00               Recall the page number
33 FRSPC?               Any room left ?
34 GTO 13               Yes, Reg X=bytes.FAT;   Reg Y=start address
```

```
35♦LBL 11                No room left in page
36 "NO ROOM"             Load "no room" message
37 GTO 14                Go tell user there's no room at the inn

38♦LBL 13                Page has some free space available
39 X<>Y                  Swap starting address of free space to Reg X
40 RDN                   Roll stack down (RegX= bytes.FAT entries)
41 FRC                   Number of free FAT entries (fraction)
42 LASTX                 Recall last value
43 INT                   Number of free bytes remaining (integer)
44 R↑                    Reg Y = free space, Reg X = program length
45 X>Y?                  Is program greater than free space ?
46 GTO 11                Yes, so go tell user of error
47 RDN                   Roll stack down
48 RDN                   Reg Y = number of labels, Reg X = freeFAT
49 X<>Y                  Swap them over
50 X<=Y?                 Is number of LBLs required less or equal to free FAT entries?
51 GTO 13                Yes, so go process it
52 "FAT OVERFLOW"        No, so load "overflow" message

53♦LBL 14                FAT overflow routine
54 XEQ 10                Go tell user he's overweight
55 GTO 12                Then go back to the beginning & get new program name

56♦LBL 13                Valid program name & free space, so proceed
57 RCL 00                Recall page number
58 CF 25                 Clear error flag to allow abort on error
59 BURNUC                Burn the program into ZEPROM
60 GTO 12                Done so get next program

61♦LBL "INIT"            Set up/initialise page routine
62 CF 19                 Not called from another program, so clear INITP flag
63 GTO 00                Go append " number" to message

64♦LBL "INITP"           Entry point if calling this programmatically
65 SF 19                 Called programmatically, so set INITP flag

66♦LBL 00                Get page number and check if blank
67 "PAGE"                Load "page" message to alpha
68 XEQ 08                Go append " number" to it & get input
69 FS? 22                Test again if input made
70 "├PAGE"               Append "page" to invalid message
71 FC? 25                Error flag still clear ?
72 GTO 14                Yes, so go tell user about error
73 STO 00                Store the page number into Reg:00
74 BLANK?                Check that the page is blank, validate input.
75 GTO 13                Invalid input or our page is blank.
76 "PG NOT BLANK"        Load "non-blank page" message
77 RCL 00                Recall page number
78 FS?C 19               Did we call this via ᵀ INITP from another program ?
79 RTN                   Yes, so assume page already initialised.
80 GTO 14                Go show error message to user

81♦LBL 13                Detect blank page or invalid input
82 FS? 25                Error flag set ?
83 GTO 13

84♦LBL 14                Tell the user about it
85 XEQ 10                Go show him the message
86 GTO 00                Then go get another page number

87♦LBL 13
```

```
 88 CF 20                  Clear bank-switched flag
 89 RCL 00                 Recall page number
 90 11.007                 Set up DSE loop counter
 91 X<Y?                   Is page number less than/equal to Reg X ?
 92 15.011                 Yes, then replace DSE loop counter in Reg X

 93♦LBL 05                 No
 94 BNKSW?                 Any of the pages bank-switched?
 95 GTO 13                 Yes, so they all will be.
 96 LASTX                  Recall page number
 97 DSE X                  Check loop counter
 98 GTO 05                 Not yet equal, so loop back

 99♦LBL 06                 Determine whether bank-switching wanted
100 "WANT BANKSW"          Load "do you want bank-switching" message
101 XEQ 09                 Go get input - needs either 0 or 1
102 "INPUT 0 OR 1"         Tell him we want '1' for bank-switched, otherwise '0'
103 FC? 25                 Test error flag  Is it still clear ?
104 GTO 14                 Yes, Go tell user about it
105  E                     Input value of 1 for testing
106 X<>Y                   Swap input and 1 around
107 X<=Y?                  Is bank-switching input less than or equal to 1 ?
108 GTO 13

109♦LBL 14                 Tell user about it
110 XEQ 10                 Go show him the message
111 GTO 06                 Go repeat the bank-switching question

112♦LBL 13                 Set up bank-switching flag & minimum FAT entries
113 X#0?                   Is bank-switching input non-zero ?
114 SF 20                  Yes, so set bank-switching flag
115 X#0?                   Is bank-switching input non-zero ?
116 6                      Yes, so set minimum FAT entries for bank-switched page
117 STO 02                 Store temporarily in Reg:02

118♦LBL 01                 Page header routine
119 "HEADER"               Load "page header" message
120 XEQ 07                 Go prompt for alpha input
121 FC? 23                 Is input flag clear ? (no alpha input?)
122 GTO 13
123 CLX                    Reg:X = 0 & disable stack lift
124 F4: 7F 00 00 00        Append three null
125 RCL 0                  }
126 X#Y?                   }  Was the alpha input greater than 11 characters ?
127 GTO 14                 Yes, so go tell user of error
128 F5: 7F 00 00 00 00     Append four more nulls
129 X<> 0                  }
130 X<> N                  }  Restore alpha register to its original content
131 STO M                  }
132 GTO 13                 Go store the header message in Reg:03 & Reg:04

133♦LBL 14                 Bad "page header" routine
134 XEQ 15                 Go load "invalid" message
135 "⊢HDR"                 Append "header" to it
136 XEQ 10                 Go show him the message
137 GTO 01                 Then go ask him for the page header again

138♦LBL 13                 Store header routine
139 ASTO 03                Store first 6 characters of header in Reg:03
140 ASHF                   Shift Alpha by 6 characters to the right
141 ASTO 04                Store the last part of the header in Reg:04
```

```
142 ISG 02                  Assume SKIP past LBL 02

143♦LBL 02                  Get trailer routine
144 "TRAILER"               Load "page trailer" message
145 XEQ 07                  Go prompt for alpha input
146 FC? 23                  Did he input anything ?
147 GTO 13                  Yes, so store it in Reg:05
148 CLX                     No, so clear Reg:X
149 F4: 7F 00 00 00         Append three nulls
150 RCL N                   }
151 X#Y?                    } Was alpha input greater than 4 characters
152 GTO 14                  Yes, so go tell user about error
153 F2: 7F 00               Append another null
154 RCL N                   }
155 X=Y?                    } Was alpha input less than 4 characters
156 GTO 14                  Yes, so go tell user about error
157 F4: 7F 00 00 00         Append three more nulls
158 CLX                     }
159 X<> N                   } Restore alpha register to its original content
160 STO M                   }
161 GTO 13                  Go store the trailer to Reg:05

162♦LBL 14                  Invalid trailer routine
163 XEQ 15                  Go load "invalid" message
164 "├TRLR"                 Append "trailer" to it
165 XEQ 10                  Go show him the message
166 GTO 02                  Then go ask him for the page trailer again

167♦LBL 13                  Store trailer into Reg:05
168 ASTO 05

169♦LBL 03                  Get XROM number routine
170 "XROM"                  Load "xrom" message
171 XEQ 08                  Go append "number ?" & get numeric input from user
172 FS? 22                  Did he input anything ?
173 "├XROM"                 Append " xrom" to invalid message in alpha
174 X#0?                    Is it non-zero ?
175 FC? 25                  Error flag still clear ?
176 GTO 14                  Go tell user about it
177 32                      Load maximum XROM-ID number
178 X>Y?                    Was his input less than 32 ?
179 GTO 13                  Yes, so proceed

180♦LBL 14                  Go tell user XROM too big
181 XEQ 10                  Go show him the message
182 GTO 03                  Then go ask him to pick another number

183♦LBL 13                  Store XROM-ID routine
184 X<>Y                    Recall XROM to Reg:X
185 STO 01                  Store XROM-ID into Reg:01

186♦LBL 04                  Get maximum number of FAT entries required
187 "NO OF LBLS "           Load "how many FAT entries wanted" message
188 XEQ 09                  Go get numeric input
189 FS? 22                  Did user input anything ?
190 "├LBLS"                 Append "labels" to invalid message
191 X#0?                    Is the input non-zero ?
192 FC? 25                  Error flag still clear ?
193 GTO 14                  Go tell user about it
194 RCL 02                  Recall minimum number of FAT entries (6 if bank-switched)
195 +                       Add minimum FAT to required number
```

```
196 65                  Load maximum possible FAT size
197 X>Y?                Is required value less than maximum ?
198 GTO 13              Yes, so continue

199♦LBL 14              Tell user about error
200 XEQ 10              Show him the message
201 GTO 04              Then go ask him for number of FAT entries again

202♦LBL 13              Process number of FAT entries
203 X<>Y                Recall FAT entries to Reg:X
204 STO 02              Store it into Reg:02
205 CLA                 Blank alpha ready for input
206 ARCL 03             Recall first part of header
207 ARCL 04             Recall second part of header
208 "⊦,"                Append "comma"
209 ARCL 05             Recall trailer
210 RCL 00              Recall page number
211 RCL 01              Recall XROM number
212 RCL 02              Recall number of FAT entries required
213  E3                 Load 1000
214 /                   Divide FAT entries by 1000
215 +                   Add to XROM-ID to give "xrom.fat"
216 FS? 20              Is this a bank-switched page ?
217 CHS                 Yes, so change the XROM sign to negative
218 CF 25               Clear error flag to allow abort on error
219 INITPG              Now initialise the page
220 FC?C 20             Is bank-switched flag clear (and clear it) ?
221 RTN                 Yes (page is not bank-switched), so return
222 RCL 00              Recall page number
223 ADDUCF              Now add the user code bank-switching stuff
224 ADDMCF              Followed by the m-code bank-switching stuff
225 RTN                 Then return

226♦LBL 07              Alpha input routine
227 "⊦ ?"               Append "question mark"
228 CF 23               Clear alpha input flag
229 AON                 Turn on Alpha ready for input
230 PROMPT              Get input from user
231 AOFF                Turn off Alpha
232 FC? 23              Didn't input anything ?
233 CLA                 No input, so clear alpha
234 RTN

235♦LBL 08
236 "⊦ NUMBER"          Append "number" to message

237♦LBL 09              Numeric input routine
238 "⊦?"                Append "question mark" to message
239 CLST                Clear the stack registers
240 CF 22               Clear flag to test for numeric input
241 PROMPT              Stop for user to key in numeric value
242 "INPUT REQ'D"       Load message telling user "input really is necessary"
243 FS? 22              Was there input ?
244 XEQ 15              Yes, so go load "invalid" message
245 CF 25               Clear error flag to allow abort on error
246 FS? 22              Was there really any input ?
247 X<0?                Yes, so test whether input was negative
248 RTN
249 INT                 Take integer of input
250 LASTX               Recall the input value to Reg X
```

```
251 X=Y?                    Was input value an integer ?
252 SF 25                   Set error trap flag
253 RTN

254♦LBL 15                  Invalid input message message routine
255 "INVALID "              Load "invalid" part of error message
256 RTN                     Go back to append the rest

257♦LBL 10                  Display routine
258 AVIEW                   Display message for the user to see
259 TONE 0                  Then wake him up
260 PSE                     Pause for him to see message
261 END
```

Bank Switching Code

The following listings detail the bank-switching code provided in the PROGRAMMER ROM. Note, however, that the functions assume the XROM number of the particular page into which they are loaded; e.g. XROM 09.

Functions	Description
BGTOxx	Bank-switched GTO from User-code.
BRTNxx	Bank-switched RTN from User-code.
BXEQxx	Bank-switched XEQ from User-code.
PBxx	Enable primary bank of XROM xx.
SBxx	Enable secondary bank of XROM xx.

Bank-switching code

This code is added into ZEPROM when the bank is initialised as being bank-switched with INITPG, or when later changed to bank-switching with ADDBSW and provides the low level ability to swap banks in that ROM page.

```
$LOAD AT FC7$
pFC7   100 ENBANK1
  xFC8 3E0 RTN
pFC9   180 ENBANK2
  xFCA 3E0 RTN
```

Functions for bank-switching in User code

The functions BGTOxx, BRTNxx and BXEQxx perform bank-switching from within user-code programs. These functions are added into ZEPROM by the ADDUCF instructions. Note that they assume the XROM number of the particular page into which they are loaded; in this case, XROM 09.

```
xFCB 0B9 9
xFCC 030 0
xFCD 00F O
xFCE 014 T
xFCF 007 G
```

```
  xFD0 002 B
BGTO09 130 LDI                  This is the post-fix byte for 'GTO IND X'.
  xFD2 073 CON      073h
  xFD3 003 JNC      godoit

  xFD4 0B9 9
  xFD5 030 0
  xFD6 011 Q
  xFD7 005 E
  xFD8 018 X
  xFD9 002 B
BXEQ09 130 LDI                  This is the post-fix byte for 'XEQ IND X'.
  xFDB 0F3 CON      0F3h
godoit 358 ST=C                 This is where 'XGI' expects it's argument.
  xFDD 100 ENBANK1              Swop banks (re-burnt as ENBANK2 for lower page)
  xFDE 31D *        XGI         Arrive here from same function, other bank.
  xFDF 092 NCGO     24C7h       Exit to mainframe, GTO or XEQ - IND X.

  xFE0 0B9 9
  xFE1 030 0
  xFE2 00E N
  xFE3 014 T
  xFE4 012 R
  xFE5 002 B
BRTN09 100 ENBANK1              Swop banks (re-burnt as ENBANK2 for lower page)
  xFE7 00D *        XRTN
  xFE8 09E NCGO     2703h       Exit to mainframe, RTN to calling User-code.
```

M-code bank-switching functions

The functions, PBxx and SBxx, enable either the primary or secondary bank of the page of XROM-ID xx. These functions are added into ZEPROM by the ADDMCF instructions. Note, however, that the functions assume the XROM number of the particular page into which they are loaded; in this case, XROM 09. Similar functions ENABLEP & ENABLES are used in the PROGRAMMER ROM to switch between primary and secondary banks of that ROM.

```
  xFE9 0B9 9
  xFEA 030 0
  xFEB 002 B
  xFEC 010 P

  xFED 003 JNC      pFC7

  xFEE 0B9 9
  xFEF 030 0
  xFF0 002 B
  xFF1 013 S

  xFF2 003 JNC      pFC9
```

Appendix C

Warranty & Servicing

Limited Warranty

Zengrange products have been developed and manufactured to high standards. Products carry a limited warranty effective from the date of original purchase.

Zeprom Voltage Converter:
Warranted against defects in materials and workmanship affecting electronic and mechanical performance for **90 days**.

ZEPROM Module:
Warranted against defects in materials and workmanship affecting electronic and mechanical performance of the module casing, connector and associated logic circuitry for **90 days**.

The ZEPROM module is supplied fully tested, with the PROGRAMMER ROM software ready programmed into one bank. On receipt, the module should be tested in an HP-41, and if it appears to be defective, returned immediately to Zengrange Ltd, or their local agent, for a free-of-charge replacement. However, once a ZEPROM module has been erased, or any location in the module has been programmed, neither Zengrange Ltd nor its agents, can accept any further responsibility due to the introduction of procedures totally beyond our control.

A ZEPROM module will normally be capable of being programmed and erased more than 100 times, but because these processes are outside our control, this cannot be guaranteed. The expected life far exceeds the frequency of reprogramming expected or intended for this type of device.

If given as a gift, the product warranty is transferred to a new owner for the remainder of that period, provided that proof of purchase date is supplied. During the warranty, we will repair or, at our option, replace a defective product, provided it is returned, shipping prepaid, together with proof of purchase to an official service representative.

Products are sold on the basis of specifications as at manufacture. There shall be no obligation to modify or update a product once manufactured.

Consumer Transactions in the United Kingdom

This warranty shall not affect the statutory rights of a consumer whose rights as Buyer and the obligations of Seller are determined by statute.

Warranty Restrictions

This warranty does not, and shall not apply if a product has been damaged by accident, misuse, modification, or service by unauthorised persons or organisations. No other expressed or implied warranty is or shall have been given. The repair, or replacement of the product is your exclusive remedy. Under no circumstances shall our liability extend to consequential or incidental damages, no matter how caused, nor shall it exceed the catalogue or sale price of the product at the time of sale. Under no circumstances shall any liability attach to us for loss or corruption of program or data material stored in any computer using our products.

Shipping for Service

During normal, or extended use, these Zengrange products will not require any maintenance. There are no user serviceable parts inside the ZEPROM module, nor in the ZVC device.

Because many factors could affect the performance of your product, including flat batteries, software corruption or the particular configuration or way in which you are using it, we recommend that you make contact with us by telephone or letter before returning a product. In this way we can help isolate any difficulties that do not require the return of a product.

In the unlikely event that the product proves to be defective, return it, postage prepaid, to:

Zengrange Ltd.,
Greenfield Road,
Leeds, W.Yorks,
England,
LS9 8DB.

Telephone:
National: Leeds (0532) 489048
International: + 44 532 489048

or to other official, local service representatives.

When returning products, be sure to include the following:

- A copy of your sales receipt, or other proof of purchase - if the warranty period has not expired.
- A description of the problem, detailing the circumstances of when and how the problem occurs.
- Details of your particular computer system - including serial number, memory configuration, ROMs plugged in, etc.

Whether or not a product is still under the warranty, it is the responsibility of the owner to ensure that the device is securely packaged to prevent damage in transit (which is not covered by our warranty) and that shipping costs to ourselves are paid.

Technical Assistance

The operating instructions in this manual are supplied with the assumption that the user has a working knowledge of the concepts, terminology, technology and equipment used. Whilst we are happy to advise on general suitability and usage of our products, the multitude of programs that could be used on our products makes it impossible for us to provide detailed technical assistance except where it relates to our own product design. Any information given, not directly relating to our products, shall be used entirely at the recipient's own risk.

POTENTIAL FOR RADIO AND TELEVISION INTERFERENCE (For U.S.A. only)

The ZVC and HP-41 generate and use radio frequency energy and, if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in subpart J of part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If your product does cause interference to radio or television reception, you are encouraged to try to correct the interference by one or more of the following measures:

- Reorientate the receiving antenna.
- Relocate the computer with respect to the receiver.
- Move the computer away from the receiver.

If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet prepared by the Federal Communication Commission helpful: How to Identify and Resolve Radio & TV Interference Problems. This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

Appendix D:

Glossary

Address ... HP-41 memory location.

Argument ... Input value required from the user. E.g. number of page to burn.

ASCII ... American Standard Code for Information Interchange (pronounced: *As'kee*). Standardised codes for letters, numbers and symbols.

Backup ... A spare copy of data kept in case the original is lost/damaged.

Bank ... 4K memory block residing within module page. In a Bank-switched module, 4K blocks can be switched on and off-line as needed.

Bank-switching ... Process by which blocks of RAM or ROM are moved into and out of the 41's addressing space as they are needed. Allows effective expansion of computer memory. See Primary & Secondary banks.

Bit ... Binary Digit. The Boolean values representing '0' or '1'.

Burning ... Process of programming data into an EPROM device.

Byte ... Basic unit of measure for computer memory or disc capacity. Usually one ASCII character takes up one byte (8-bits) of memory.

Configuration ... Organisation or set up of memory or peripherals.

Core ... A 4K block of memory in ZEPROM or HP-ROM module.

Crash ... Trauma suffered by the HP-41 resulting in temporary loss of control and/or data held in memory. May be caused by keying in wrong input (in writing M-code), wrongly burning code to interrupt locations in ZEPROM, from static-discharge, etc.

Default ... A value or selection made by the program in lieu of operator input.

Directory ... A list (catalogue) of files on a disc or cassette tape.

EPROM ... Erasable & Programmable Read Only Memory. Integrated Circuit (re)programmable with software for insertion into an HP-41 port.

File ... A collection of related information, such as a program stored on cassette or disc.

Filename ... Unique name given to a file saved on cassette or disc.

Function Address Table ... FAT. List of pointers to functions/programs in a ROM.

Hardware ... Electronic equipment (disc drives, display, etc) making up a computer.

HP-IL ... Hewlett-Packard Interface Loop. System for interconnecting low, battery powered devices such as disc drives or displays on HP-41.

I/O Ports ... Slots at the rear of HP-41 into which modules and other accessories are plugged.

Initialise Process a ZEPROM to allow data/programs to be stored. Until initialised ZEPROM doesn't contain data necessary to be recognised by HP-41. Also applied to cassette/disc media.

Kilobyte Thousand bytes (actually 1024 bytes). Also known as 'Kb' or 'K'.

Label LBL. Point in a program to which the 41 can branch to.

Machine Code The native, low-level language of the HP-41's processor (CPU).

Main Memory The area of the HP-41 memory used for register and program storage.

Mass storage medium A cassette or disc used in an HP-IL device such as a Cassette or Disc drive.. Also called a flexible, floppy or micro-disc or diskette.

Memory cores See Core.

Module Plug-in case containing RAM or ROM chip for connecting to HP-41.

Off-line Disabled (invisible) to the HP-41. Data cannot be accessed until device or memory is once again On-line. See On-line.

On-line Enabled (visible) to the HP-41. Data can only be accessed from On-line storage. See Off-line.

Operating System Software built into the HP-41 that controls operation.

Page header Identifying ROM name normally stored at the start of a ROM page.

Page trailer 4-character revision code stored at the end of all ROM cores.

Parameter Input value required from the user. E.g. number of page to burn.

Primary bank 4K-bank in a bank-switched page that is enabled by the HP-41 by default. See Secondary bank.

Q-ROM Quasi-ROM device, also called RAM Storage Unit. Large, battery backed-up RAM expansions that simulate 41 ROMs. Can be used to develop M-Code.

RAM Random Access Memory. (More correctly: Read & Write Memory.) Integrated circuits (chips) making up a computer's volatile memory.

ROM Read Only Memory. Non-volatile chip programmed with software for insertion into an HP-41 port.

Secondary bank Alternative 4K-bank in a bank-switched page that is can be swapped (enabled) in place of Primary bank as desired.

Software Programs used by the HP-41. E.g. Zenrom, Advantage, Maths Pac.

Synthetic Programming Technique for creating user code instructions unintended by HP and which the user cannot key-in directly.

User Code Programs written using Reverse Polish Notation (RPN) instructions.

Utility An extra program, supplied with an application, but separate from it, that provides extensions to the application.

Word Logical grouping of bits. In the case of the HP-41, a word is 10 bits.

XROM number An identifying number assigned to a ROM page.

INDEX