PROMQUEEN/64
OWNERS MANUAL

GLOUCESTER COMPUTER CO. Inc.

THE PROMQUEEN/64 MANUAL

Gloucester Computer Co. Inc.
1 Blackburn Center
Gloucester, MA 01930
USER HOT LINE: 617-283-7719

PROPRIETARY NOTICE

The information and product design as disclosed in this manual as they pertain to the PROMQUEEN 64 TM Romware development cartridge and the Autohex/64 TM software were originated by and are the property of Gloucester Computer Co., Inc.

The contents of this manual have been reviewed for accuracy; however, no responsibility is assumed should any portion be inaccurate.

Specifications and information herein are subject to change to allow for the introduction of software and hardware upgrades.

Commodore 64 is a trademark of Commodore Business Machines, Inc.

INTRODUCTION

Thank you for your purchase of our PROMQUEEN 64 ROM software development system. The cartridge and its software are designed for versatility, so you must bear with us as we attempt to acquaint you with its many features. To use the cartridge properly you must understand three categories of information, which are:

1. The three modes of the cartridge controlled by the 3-position toggle switch.

2. Selection of EPROM type by appropriate setting of the matrix switch.

3. Use of the software for manipulation of hexadecimal lists and communications to and from other devices such as the printer, RS 232 interface, cassette recorder, and disk drive.

You are assumed to be familiar with number bases and the hexadecimal system, which is most natural for microprocessor work. Please read pages 215-217 of the Commodore 64 Programmer's Reference Guide if you are unfamiliar with this.

This manual thus consists of four parts, including an inevitable catch-all section for miscellaneous information. It is written in tutorial style to give you hands on experience with each capability of the cartridge. Please take the time to work along with the exercises as they are given. You'll gain a thorough understanding of the cartridge as well as ability to use it creatively. You'll also save on wasted time or telephone calls.

SO PLEASE READ THIS MANUAL THOROUGHLY

...and if all else fails, our user hot line is 617-283-7719

TABLE OF CONTENTS

SECTION 4 ; APPENDICES

ILLUSTRATIONS:

What is an EPROM?

All computer programs are stored for use within a computer as lists of numbers in the memory space of the processing unit that executes them. Any given processor, such as the 6510 chip in the C-64 can address a certain number of memory locations. These are numbered in binary as far as the processing unit is concerned, so it can electrically specify any particular address among all the memory locations available. The 6510, for instance, can identify by means of patterns of 1's and 0's on its 16 address lines any of 65536 different addresses, and read or possiby change the contents thereof. Each address contains an 8-bit binary number. If we express the possible range of numbers that can be expressed in 8 binary digits in other number bases, the range is 0 to 255 in decimal, or 0-FF in the base 16 hexadecimal system. Thus an "8 bit" computer program appears in computer memory as a list of numbers, all of which fall in the range from 0-255, or $0 to $FF.

If the computer's memory is RAM, the program must be loaded before it can be run. RAM memory contents are subject to change, and loss of power causes total loss of RAM contents. Hence we have disk drives and cassettes, to store our programs in a more permanent, yet reloadable form.

It isn't always advantageous to use disks or cassettes for program storage. Perhaps we have dedicated the computer to a particular task and wish to avoid the expense. Perhaps the application is rugged and we can't expect mechanical devices to hold up.

As an alternative we can take the lists of numbers that comprise our programs and store them on chips that are made to plug in directly to the memory space of our processor. EPROMs are made just for this purpose. You can take a list of binary numbers in the range from 0-255 in decimal or $0-$FF and "burn" it on a chip that just plugs right in. The program becomes built in to the computer, ready to run as soon as power is applied. EPROM's are also erasable, if you want to use them for a different program, they need only to be exposed to a mercury vapor lamp for a period of time. In addition, your PROMQUEEN 64 allows you to program and erase some EEPROM's, which you can erase electrically as well as program from the PROMQUEEN 64.

The PROMQUEEN 64 gives you a variety of tools to handle both the lists of numbers and the chips that you can build into your computers.

SECTION 1 : OPERATING MODES

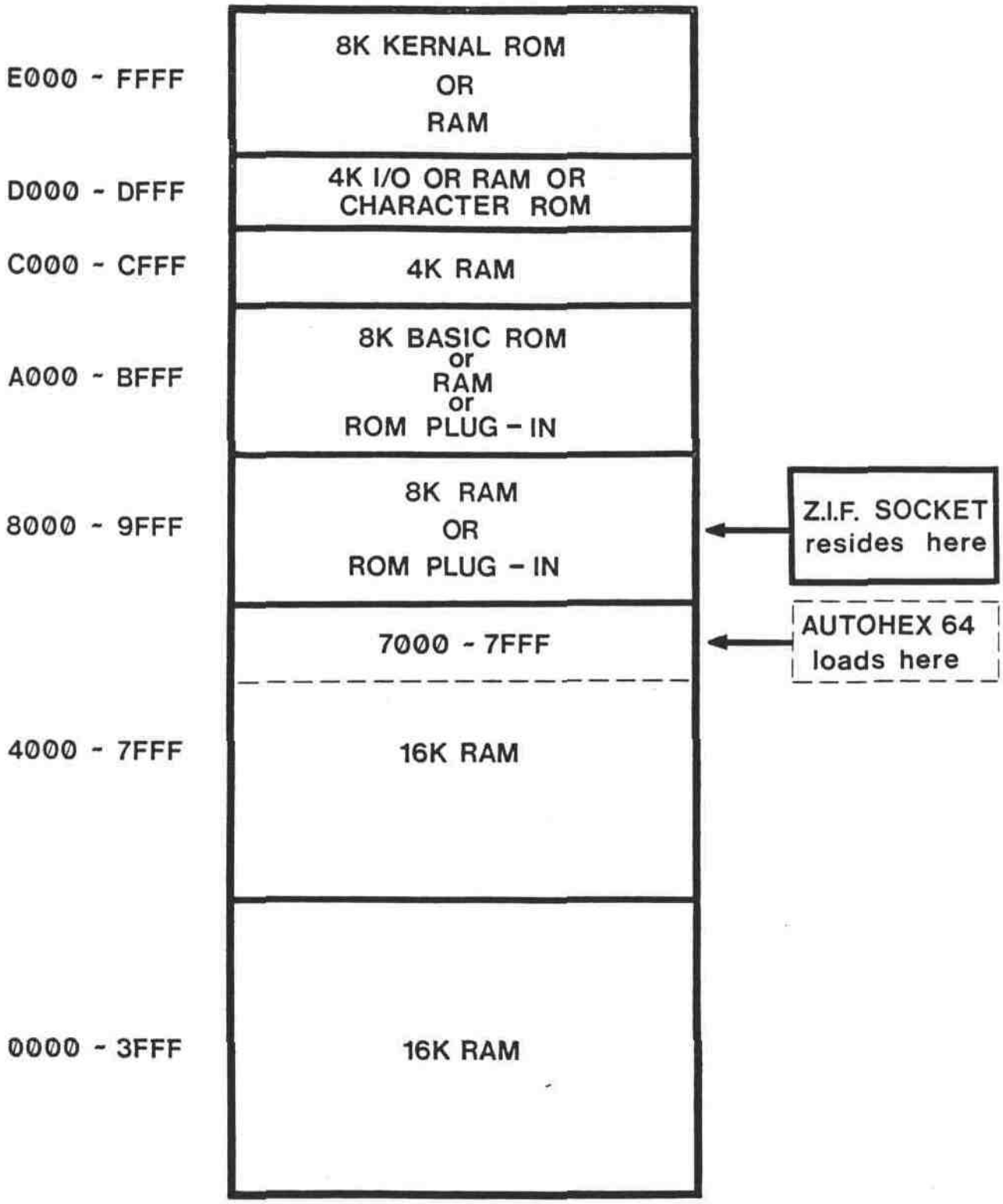
1.1 C-64 Memory Map Options

The C-64 allows for several different memory configurations depending upon the signal levels present on 2 external inputs in the expansion port as well as the setting of a bank control register at address 1 in C-64 memory. These signals interact in a variety of ways which are covered in the C-64 programmer's reference guide, pages 260-267. Not all of the available memory maps can be used with the PQ-64. The cartridge is designed to control the C-64 memory map automatically. If you manually intervene to change the bank selection register internal to the C-64 the cartridge may not work properly. Specifically you should avoid switching out the BASIC interpreter or the Kernal ROM.

Of all the potential memory maps of the C-64, only two are used by the cartridge and its software. The cartridge will remove itself entirely from the C-64 memory map when the three position toggle is in the "0" position (toward you). This allows you to work with the 8k of RAM from $8000 (32768) to $9FFF (40959) normally present in the C-64. When you set the toggle to C (copy) or B (burn) this internal RAM is banked out and replaced with whatever ROM chip is installed in the ZIP socket. If the ROM chip is 8k long or longer, it will entirely fill the range of addresses from $8000 to 9FFF. If it is smaller than 8k, it will appear more than once. A 4k ROM for instance appears both at $8000 and at $9000. A 2k ROM appears 4 times, a 1k ROM appears 8 times. Any ROM can be accessed wherever it appears in the C-64 memory, but it won't appear at all unless the toggle switch is set to C or B. ROMs are always addressed in the range from $8000 to $9FFF for reading, and the EPROM programmer algorithm expects them to appear in this range as well.

If you are strictly using the cartridge hardware only, you can use the normal C-64 RAM from $800 to $7FFF as RAM workspace and utility programming area. You can also use RAM at $COOO-$CFFF. The autohex 64 software, when used, takes up to 4k of memory from $7000 to $7FFF. Loading Autohex 64 can effect the memory adjustments for the C-64 operating system, and is covered in detail in Section 3.1.

64 MEMORY MAP



1.2 The Cartridge Control Switch

Before you plug in the cartridge for the first time, get the feel of this switch. Do not change any settings on the matrix switch. Notice how the EPROM that came with the cartridge is installed notched end and open pins to the left. Lift up on the ZIP socket handle to release the Aufcohex EPROM and lift it out of the socket. Set the EPROM on conductive foam. Set the control switch to the "0" position, towards the card edge connector end of the cartridge. Turn off your C-64, and plug in the cartridge to the C-64 expansion port. ALWAYS TURN OFF THE C-64 WHEN INSTALLING OR REMOVING CARTRIDGES. The control switch need not be set to "0" when turning on the C-64, but you should avoid it being set to "B" as this could inadvertently program an EPROM in the ZIF socket on power up.

1.3 "Out" Mode

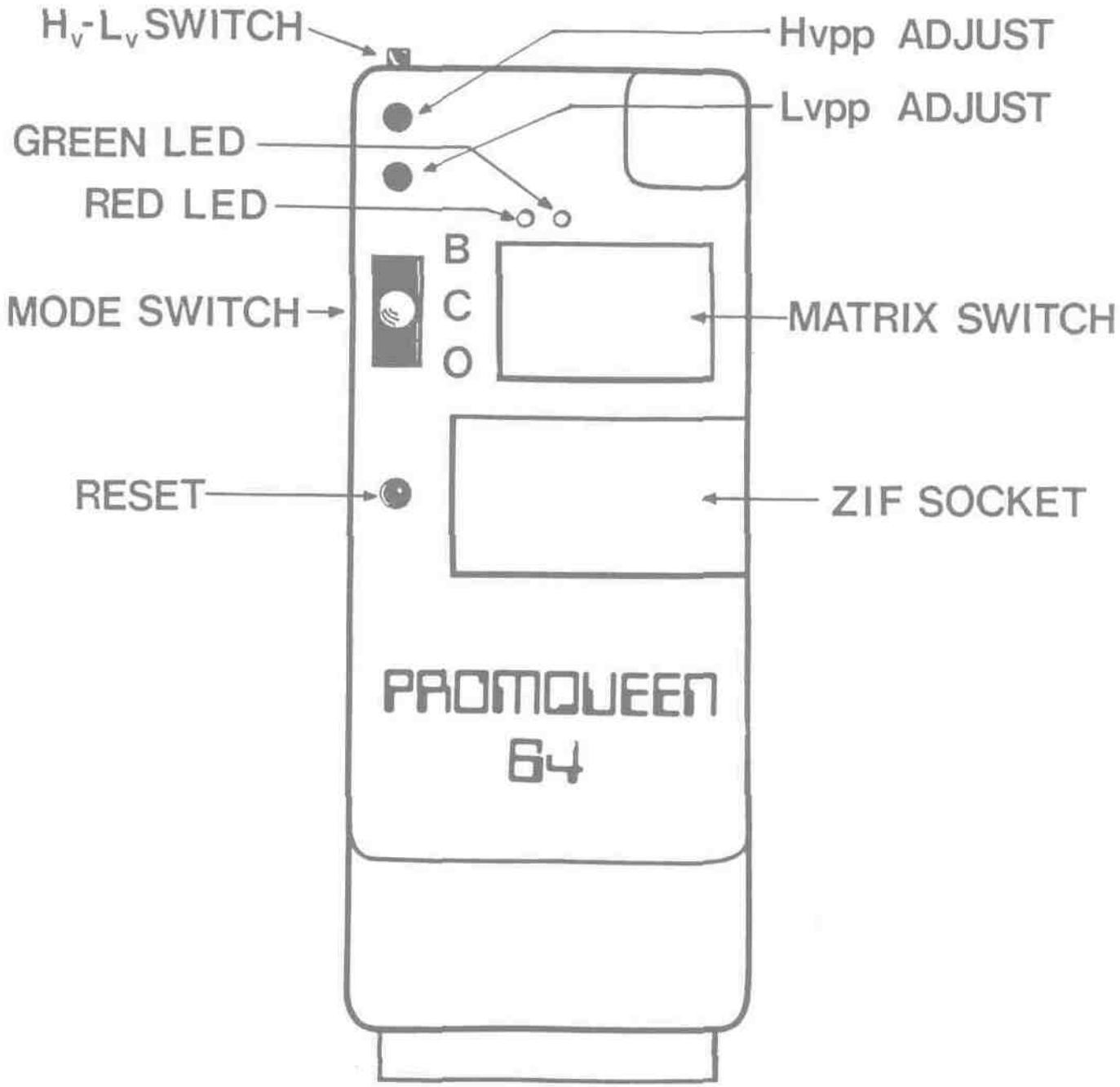
Check again that the mode switch is set to "O", and that the cartridge is firmly seated in the expansion port. Turn on the C-64. Notice that the startup message says 38911 bytes free. This is the cartridge "OUT" mode. The C-64 does not know that the cartridge is present. It is using internal RAM to fill the address space from $8000 to $9FFF, and this RAM is available for BASIC.

Note that both LED's on the cartridge are out. This signals that the ZIF socket and MATRIX switch on the PROMQUEEN 64 are de-energized. It would now be safe to install or remove EPROMs from the ZIF socket, or to adjust the MATRIX switch for different types of EPROM. Leave the C-64 on.

1.4 Copy Mode

With no EPROMs in the socket, set the mode switch to the center "C" position. Notice that the green LED comes on. This tells you the ZIF socket and MATRIX switch are energized. NEVER CHANGE EPROMs OR MATRIX SWITCH SETTINGS WHEN THE GREEN LED IS ON. Now leave the switch in "C" position, but turn off the C-64. Turn the C-64 on again. Notice that the startup message now says 30719 bytes free. The C-64 notices the cartridge when the mode is set to "C", and adjusts itself to address the cartridge ROM, instead of internal RAM. Actually this happens the instant you set the cartridge to "COPY" mode BUT the software operating system has no way of detecting this, except during a RESET procedure. Hence, unless you have

PQ-64 OPERATOR CONTROLS



first set the top of memory pointers below $8000, some of BASIC programs may dissappear when the cartridge is set to **"C",** assuming a BASIC program was previously loaded to the C-64.

Now turn the toggle switch back to "O", watch the green light go out, and re-install the Autohex 64 EPROM, notched end to the left, unused pins on the socket to the left, as it was shipped to you. Get the feel of the ZIF socket. Notice that it is released and ready to change chips when the handle is up, and that chips are latched down when the handle is down. Be sure the EPROM sits properly in the socket and latch it down. Turn the toggle switch back to "C" and hit the reset button. You should see a menu on a white screen after 2-3 seconds delay. Type an X on the keyboard to get out of this program. READY and a blinking cursor should appear. Now type this line of BASIC

PRINT FRE(O) <RETURN> You should see

26621 bytes, free for BASIC now.

The Autohex 64 software is an example of an auto start program. It automatically runs (provided it is installed in memory at $8000) as a result of any new system reset. When Autohex 64 loads itself to memory from a RESET it adjusts BASIC and operating system pointers to limit BASIC to memory below $7000. This protects Autohex from BASIC, and lets you run BASIC programs with Autohex in memory without damage to Autohex code. The additional chunk of missing memory from $7000 to $7FFF holds the Autohex program when it is loaded into the C/64. Autohex copies itself to $7000-$7FFF as soon as it is called, so the EPROM on which it comes can be removed. Turn the socket back to "0" and type SYS 29181 <RETURN> to restart the Autohex program. Notice that the menu still appears, even though the EPROM is outside of memory.

Turn the mode toggle back to "C", and then turn the C-64 off, then on again. Notice that the software auto starts this way also. Turn the toggle back off, and remove the Autohex EPROM. Turn the toggle back to "C". The green LED should be on, no EPROM in the ZIF socket.

1.5 Burn Mode

The third and final mode, "B" sets the cartridge for burning EPROMs. The software will prompt you to set the

switch to "B" when appropriate. To see how this works, type a B to the Autohex/64 Menu. The screen will come back verifying that you have asked to burn an EPROM. Don't install one now. lets work through an example "dry".

On the screen you see the prompt "FROM" and a blinking cursor. You are being asked where in the memory of the C-64 is the code you want to burn on EPROM. If we want to copy onto EPROM say, the first 1k of code from the C-64 BASIC interpreter, we should give the start address of ?AOOO. Type a $ sign, an A, 3 zeroes, and hit return (there are other ways to specify addresses, we will get to this) . Now you should see also a "TO" and a cursor. Type a $A3FF (that is a $ sign signifying HEX, then an A, a three, and two Fs, and a <RETURN>. An empty ZIF socket looks to Autohex the same as an erased EPROM, so the screen will have turned green, with the message "SET SWITCH TO BURN" displayed. Now is when you set the switch to the furthest position, "B" mode. Do so, and notice the RED LED come on, confirming the cartridge's readiness to burn. AFTER switching to burn, hit the <RETURN> key. Notice the red display and the running count of the address being burned on the screen. This length of burn will take about 50 seconds. When it's done, the screen will turn blue and tell you to SET SWITCH TO COPY. Set the toggle back to the center position and hit a <RETURN>. Since we didn't burn an EPROM, we get error messages each time we hit <RETURN>. Type an X to end the verification, and a second X to get back to the menu.

As long as you use Autohex 64 to burn EPROMs, you will be prompted to set to and from BURN mode whenever necessary. You can also use your own software to burn EPROMs. This is explained later.

IMPORTANT: Never burn past the end of an EPROM. If the EPROM is shorter than 8k it appears more than once in the 8k space occupied by the PROMQUEEN 64. Thus you could be re-burning the first part of the EPROM with FALSE data if you burn past the end. Look ahead to page 15 to find a diagram that depicts how Autohex maps code from C-64 RAM to EPROM when BURNING.

SECTION 2 : EPROM TYPE SELECTION

2.1 The. Matrix Switch

The PROMQUEEN 64 allows you to deal with a wide range of ROM devices, either for reading, writing or both. Instead of "personality modules" the cartridge has a Matrix switch to allow for a variety of pinouts and signal configurations. Applicable signals are connected to appropriate pins through the MATRIX switch settings. You will find that the arrangement adapts to many different 5 volt JEDEC pinout devices.

The PROMQUEEN 64 is shipped with the MATRIX switch set for 2732s, on which the Autohex 64 software is supplied. You must use this setting to load the Autohex software. We will shortly discuss how to copy Autohex on other EPROMs, as well as disc and cassette.

The MATRIX switch has 10 inputs and 8 outputs. Two of the outputs are "recursive" in that they re-appear as inputs, but multiplexed with other signals through the MODE switch. To assist you in devising settings of the MATRIX switch for devices other than those tabulated in the MATRIX Switch Settings Guide (section 2.3), FIG 2 illustrates the signals into and out of the MATRIX switch.

Note the symbol under the ZIP socket on the PROMQUEEN 64 case. It denotes how EPROMs are to be installed. The ZIF socket has 28 pins, so be sure that 24 pin devices get installed so that all unused socket pin positions are on the left. All devices, 24 pin or 28 pin are installed with the notched end facing left, and the other end all the way to the right of the socket.

It is always safe to change MATRIX switch settings when the MODE switch is set to "0" and the green light is out. DON'T CHANGE MATRIX SWITCH SETTINGS IN THE OTHER TWO MODES, just set the toggle to "0" first, then change. If you want to make one 8k EPROM out of 8-lk EPROMs, just set the MATRIX for the Iks first, copy each in succession to RAM (the Autohex software does this handily) , then readjust for the 8k EPROM, and burn the code.

NOTE: USE A TOOTHPICK, NAIL OR OTHER CLEAN POINTED OBJECT TO ADJUST THE MATRIX SWITCH. DO NOT USE PENCILS OR OTHER OBJECTS THAT LEAVE RESIDUES - THESE MAY FOUL THE SWITCH. ALWAYS LEAVE THE MATRIX SET TO A VALID EPROM CONFIGURATION.

MATRIX I/O DIAGRAM

INCOMING SIGNALS AND VOLTAGES

PIN	BURN	COPY	PIN
1	⌐¬ PULSE	HIGH	
	HIGH	LOW	2
3	¬⌐ PULSE	LOW	
	NOT USED	NOT USED	4
5	A12	A12	
	Dout	CE	6
7	AH	AH	
	+V	+V	8
9	Vpp	Cout	
	Gnd	Gnd	0

OUTGOING CONNECTIONS

A NOT PRESENT
B PIN 26 of ZIF SOCKET
(Cout) C SIGNAL to Vpp PIN in COPY MODE
(Dout) D SIGNAL to CE PIN in BURN MODE
E PIN 2 of ZIF SOCKET
F PIN 27 of ZIF SOCKET
G PIN 23 of ZIF SOCKET
H PIN 22 of ZIF SOCKET
J PIN 20 of ZIF SOCKET
K NOT PRESENT

2.2 Adjustinq the programming voltage

EPROMs are most commonly programmed with the programming potential set to 25 volts. However, faster and larger devices, as well as some EEPROMs, require a lower programming voltage, 21 volts or less. To accommodate these devices, the programming voltage power supply on the PROMQUEEN 64 is adjustable down to 6-7 volts.

For convience the cartridge is equipped with a switch on its far end that allows you to select from two preset voltages. The cartridge is shipped with the Low Voltage option set to 21 Volts and the High Voltage option set to 25 Volts. These two programming voltages cover the devices tabulated in the MATRIX selector guide. Set the switch to low for Vpp = 21 Volt devices, set the switch to high for Vpp = 25 Volt devices.

Two potentiometers control the programming voltage power supply. These are accessed through two labeled holes in the top of the case. In ordinary use it should not be necessary to readjust them. If you want to check the accuracy of the settings or change to a special voltage, the voltage is available on the ZIP socket for measurement. Remove any EPROMS from the ZIP socket, set the cartridge to BURN (the red LED should come on) , and measure the voltage between pin 1 and 14. Pin 1 is the leftmost pin on the bottom pin row of the ZIF socket. Use the postive lead of a 10,000 ohm/volt or greater VOM on pin 1. Pin 14, which is ground, is the rightmost pin on the bottom row. Adjust the appropiate pot with a small flat bladed screwdriver. Be sure that the pot you try to adjust is the one selected by the programming voltage selector switch.

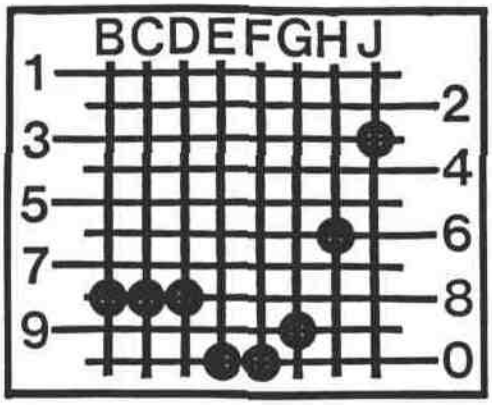
2.3 MATRIX switch settings for common EPROMs

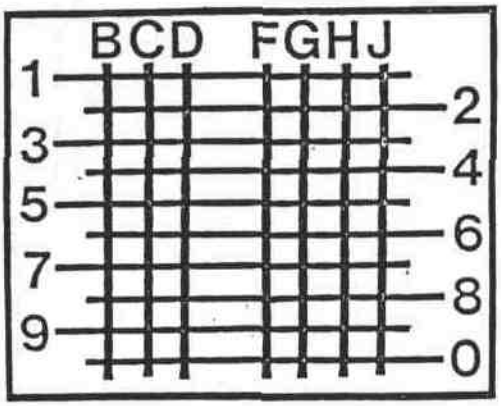
For your convenience MATRIX switch settings and programming voltages for a variety of devices is presented graphically on the following pages.

Note that the PROMQUEEN 64 handles EPROMs that are larger than 8K in 8K blocks. Matrix settings for these devices are shown for each 8K block. You must adjust the matrix to select the active 8K on the chip.

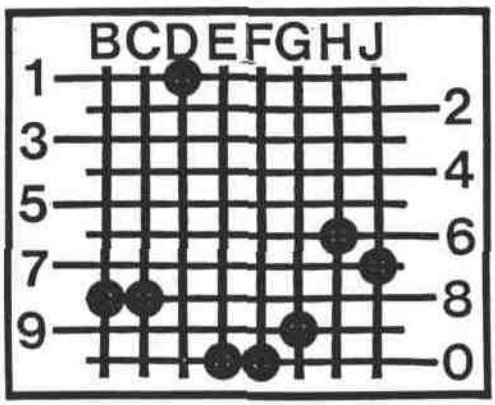
2500 MATRIX GUIDE

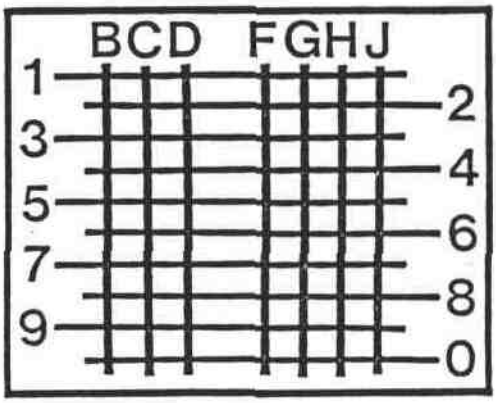
2516 (Vpp,25v.)



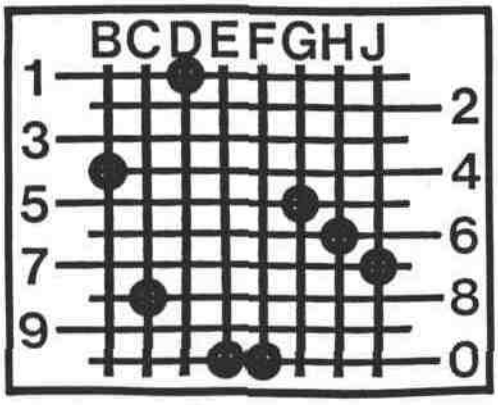


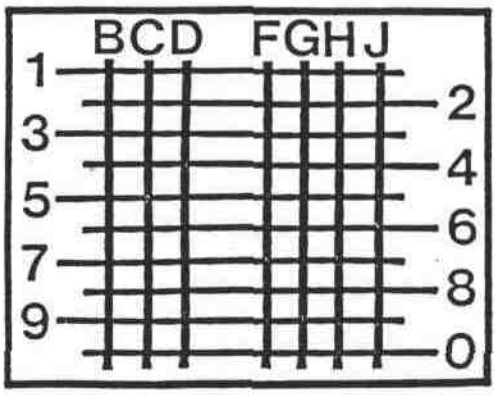
2532 (Vpp, 25v.)





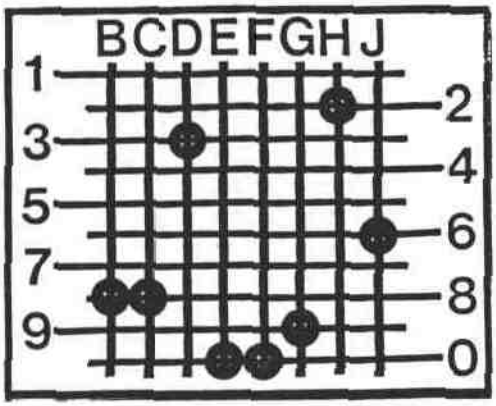
2564 (Vpp,25v.)



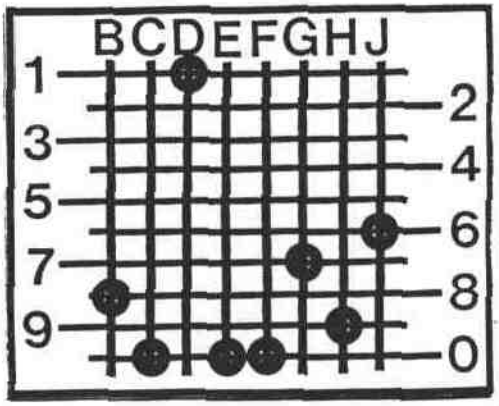


2700 MATRIX GUIDE

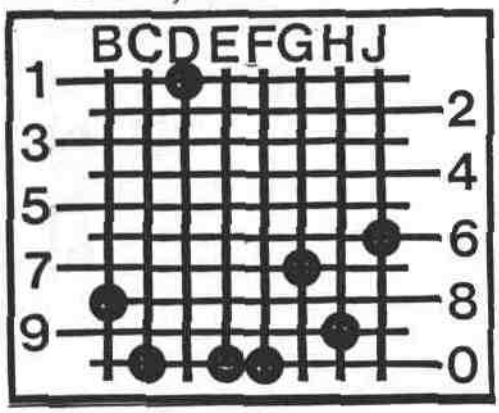
27C16, 2716 (Vpp, 25v.)

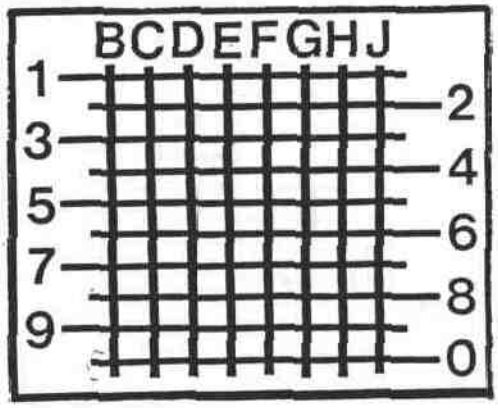


2732A (Vpp, 21v.)

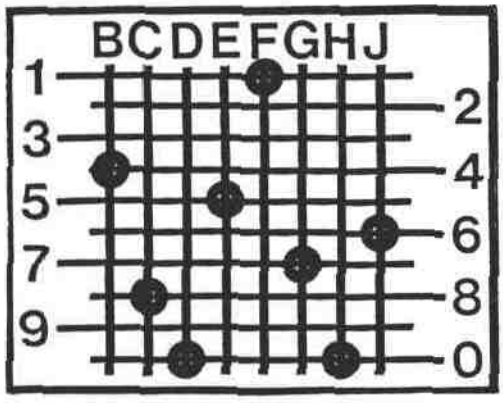


27C32, 2732 (Vpp, 25v.)

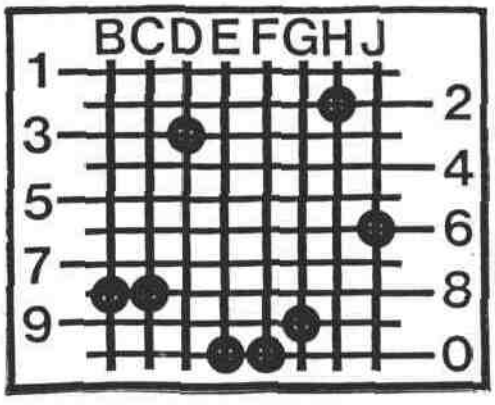




27C64, 2764 (Vpp, 21v.)

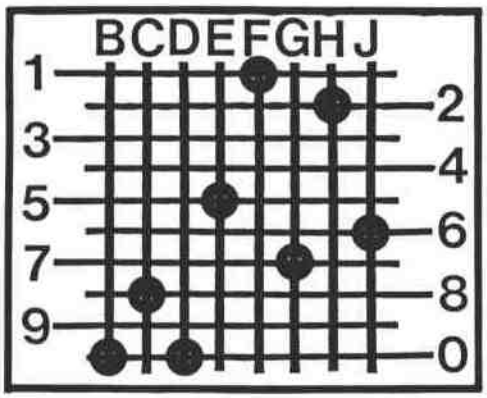


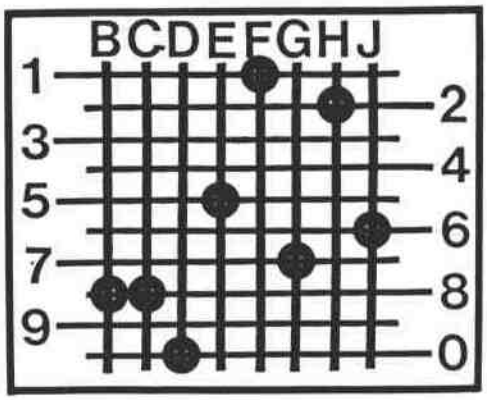
2758AorB (Vpp, 25v.)



27128

FIRST 8K (Vpp-21v) SECOND 8K

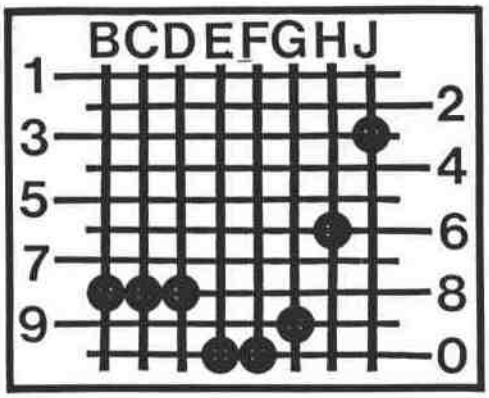


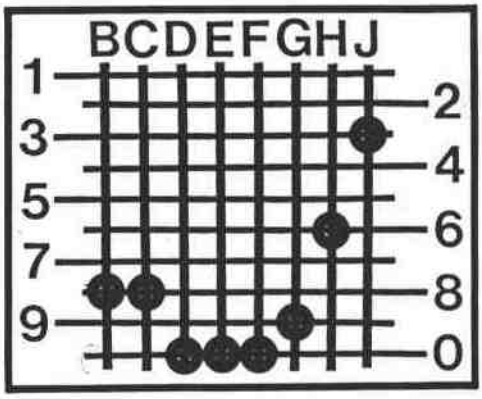


EEPROMS

48016

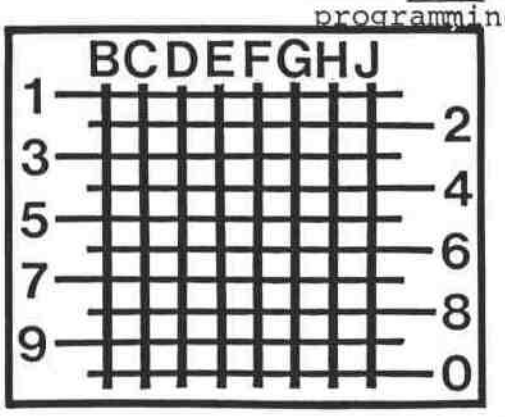
BURN (Vpp-25v) ERASE

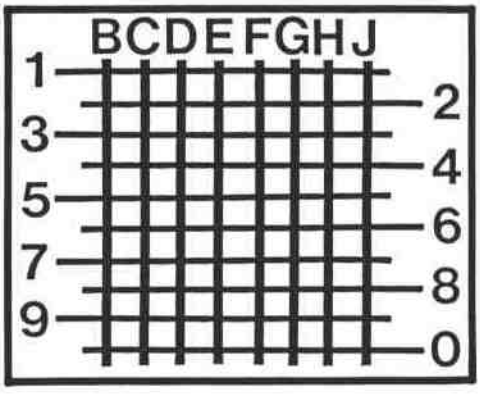




NOTES:

1) Slider D must be set to +V to program, or ground to erase
2) To erase, set Matrix switch as shown and burn only the first 4 addresses as though programming.





SECTION 3 : THE AUTOHEX 64 SOFTWARE

3.1 Loading the software from EPROM

The easiest way to load the Autohex/64 software is simply to leave its EPROM in the ZIF socket, the MODE switch set to "C" (the middle position), and turn on the C-64. The software auto loads and runs. (Be sure the MATRIX is set for 2732s first.)

If you install the Autohex EPROM on the PROMQUEEN 64 in a warm C-64, turn the MODE switch to "C", and hit the RESET button on the PROMQUEEN 64, the software will load and run the same as from a cold start.

The RESET button causes a cold start to be executed. This does not change most memory contents, but does execute a "NEW" command through BASIC, so BASIC programs in memory will remain there, but won't RUN or LIST.

When Autohex is cold loaded from EPROM, either by power up or RESET, it automatically sets the top of memory and top of BASIC pointers to $7000, protecting itself both from BASIC and its own use of RS-232 buffer space. It does not protect BASIC from the RS-232 buffers however, so if you want to use the Autohex RS-232 in conjunction with BASIC programs, be sure to set the two top of BASIC pointers to $6DOO before running BASIC. The RS-232 buffers will use the space between $6DOO to $7000.

There are two additional ways to warm load the Autohex program from EPROM through software. You can do a software RESET by giving the command SYS 64738 <RETURN>. This has the same effect as a hardware RESET. In addition, you can soft load it from EPROM with the command SYS 32813. If you soft load this way the BASIC pointers will remain undisturbed, only the KERNAL top of memory will be set to protect Autohex against its own use of RS232 buffers. Please be sure and set the pointers to protect BASIC if you need to.

If you want to return to the normal C-64 operating system from Autohex/64, just type an "X" to the main menu prompt. Then, to restart the Autohex, now that its already loaded, use the BASIC command SYS 29181 <RETURN>.

REMEMBER

COLD LOAD Autohex with the EPROM in the ZIF socket MODE switch to "C" by turning on the C-64 or hitting the PROMQUEEN 64 RESET button.

WARM LOAD from EPROM with SYS 32813 <RETURN>. Adjust BASIC pointers ($33-$34 and $37-$38) if you'll be running BASIC.

RESTART Autohex after escape to BASIC with SYS 29181 <RETURN>.

ESCAPE TO BASIC by typing an "X" when the main menu is displayed.

3.2 Main Menu Options

The Main Menu lists the 5 general categories of activity you can carry out with the software.

The EDIT HEX option, obtained by typing an A, is a kit of tools for editing, moving and patching pieces of code in C-64 memory. It gives you 20 keystroke commands which you can use to work on memory contents anywhere in C-64 RAM. These are covered in detail in the software tutorial section.

The BURN EPROM option, obtained by typing a B, is used to program EPROMs, for verifications, for erase checking, etc.

The SAVE ON DEV. option allows code to be sent to various devices from C-64 memory. Allowed devices include:

Dev #1	Cassette
Dev #2	RS232 Port
Dev #4	Printers
Dev #5	Printers
Dev #6	Disc Drives (numbered in hexadecimal) TO
Dev #F	Disc Drives (numbered in hexadecimal)

The LOAD FRM DEV option allows you to put CODE into designated memory from cassette, discs numbered from 6 to F, and RS232 interface.

The V'Fy SAVE option, obtained by typing an E, lets you check files saved to TAPE or DISC against the original file in memory.

These general features will be worked through in the following sections.

3.3 Copying EPROMs

It's a good idea to make a back-up copy of Autohex. Let us use the Autohex to save itself several different

ways. First get the Autohex into the 64. If it isn't already loaded, turn the MODE to "O", install the Autohex EPROM on the socket, notch and open pins on the left, turn the MODE switch to C, and hit the RESET button. Leave the EPROM in the socket and the mode set to C.

3.3.1 Copy to another EPROM

Now that Autohex is already in RAM, it could be burned onto EPROM. The only thing you need to look out for relates to how Autohex decides where to burn on EPROM the data it fetches from 64 memory. The cartridge itself is an 8k block of memory. It fetches data from one of 5 possible 8k blocks.

The possible 8k blocks are:

```
  0000-1FFF
  2000-3FFF
  4000-5FFF
  6000-7FFF
* AOOO-BFFF
```

* only with BASIC ROM banked out

The other blocks are unavailable either because they are already occupied by the PROMQUEEN 64 or would require the I/O or KERNAL ROM areas to be banked, without which Autohex will not run.

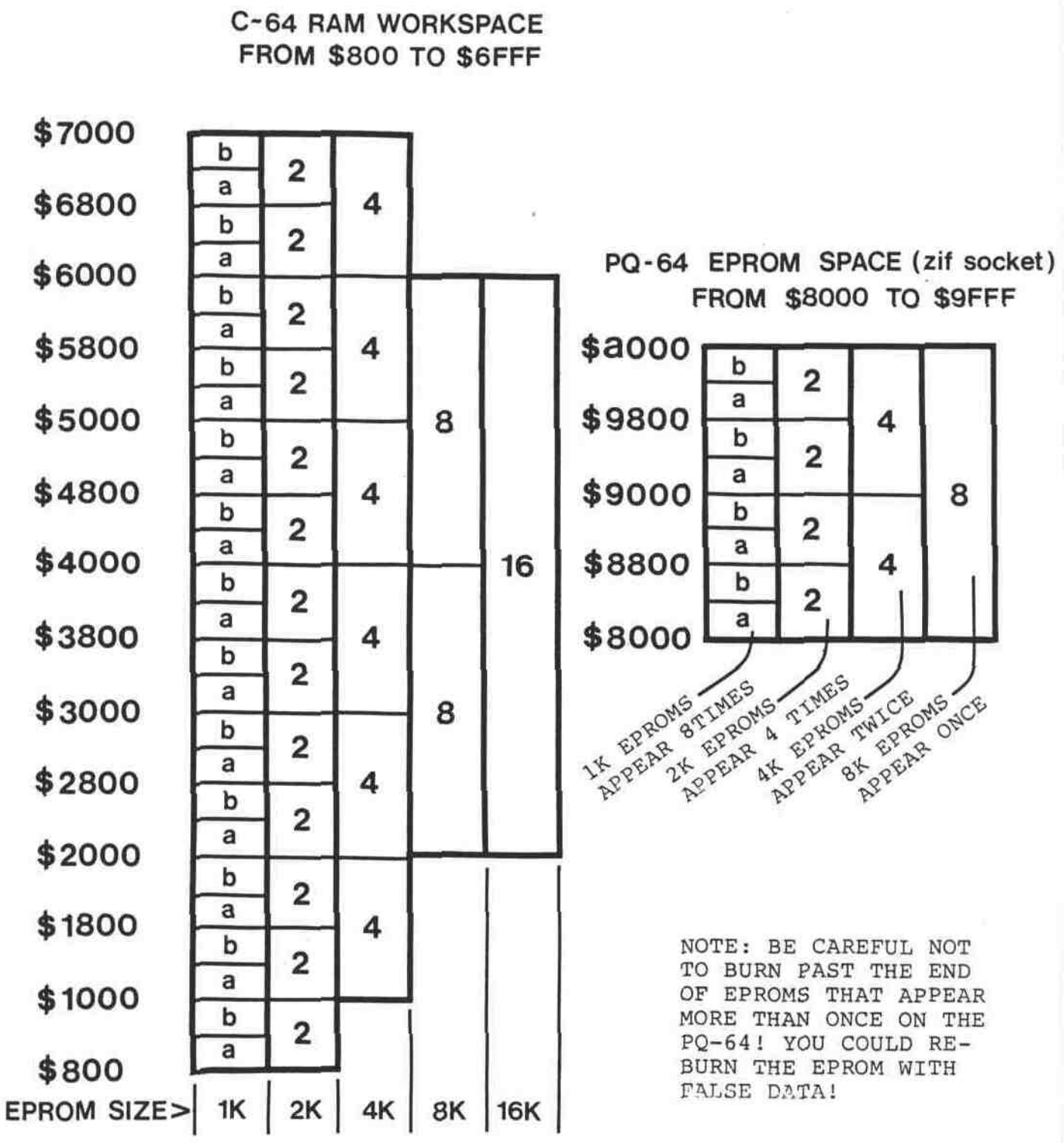
The code is burned from its location within its 8k source block to the corresponding location in the 8k of EPROM space on the PROMQUEEN 64. Hence the byte at, for instance, $4013 would be burned onto the $13th memory location on an 8k EPROM. A byte at $5160 would be burned onto the $1160th memory location on an 8k EPROM. See the diagram on page 15 for more detail on how this works on variously sized EPROMs.

If one is burning 4k EPROMs, the EPROM appears twice in the 8k PROMQUEEN 64, so it isn't necessary to move the code, it will burn as well in the upper half of the PQ's 8k as it will in the lower.

So we can just go ahead and burn the Autohex onto another 2732. If we were burning it on a 2764, we'd want it to appear in the first 4k, so the code comprising Autohex would have to be moved to $2000, $4000 or $6000. Transfers like this are sometimes needed.

Let us copy Autohex direct from its EPROM to 64 RAM to illustrate how any other EPROM is copied.

WHERE TO MAP CODE IN C-64 WORKSPACE



NOTE: BE CAREFUL NOT TO BURN PAST THE END OF EPROMS THAT APPEAR MORE THAN ONCE ON THE PQ-64! YOU COULD RE-BURN THE EPROM WITH FALSE DATA!

* EPROMS LARGER THAN 8K CAN ONLY BE READ OR PROGRAMMED 8K AT A TIME. USE MATRIX SWITCH TO SELECT WHICH 8K BY SETTING/CLEARING THE HIGHEST ORDER ADDRESS LINE(S).

\+ USE A SECTIONS TO READ/PROGRAM 2758As.USE B SECTIONS TO READ/PROGRAM 2758Bs.

To copy EPROMs into RAM you must use the transfer function in the EDIT HEX package. Type an "A" to select EDIT HEX from the Main Menu.

Now you will see that you are asked where in memory you want to Edit. Let us pick $4000 as a suitable even numbered 8k boundary. Type

```
$4000 <RETURN>
```

Now you will see 15 lines of display and a blinking cursor on the line numbered 4000 (decimal 16384). Now we can call the transfer by typing a T. The prompt "XFER FRM" and a blinking cursor will appear at the lower left of the screen. The EPROM on the PROMQUEEN 64 starts at $8000, so type $8000 <RETURN>. Now you will see another prompt:

```
STOP @ 0000
```

NOTE: No dollar or number sign is used with this STOP 0000 Prompt!

cursor blinking on the first 0. Autohex is a 4 kilobyte program, so we need to define the end of our RAM workspace at 4FFF. Change the 0's to 4FFF and hit a <RETURN>. Now the prompts have disappeared and you should see the start of Autohex code has appeared on the display. If so, the transfer was successful.

Now we just burn the new EPROM. Change the mode switch to 0. Remove the Autohex EPROM and replace it with a clear 2732 type EPROM. Put the MODE switch back in C position. Type an X to get back to the main menu. Type a B to select the BURN EPROM routine. The prompt FROM appears, and the cursor. Since we put the code at $4000, we now type $4000 and hit a <RETURN>. Now you are prompted with a TO. Since our code ends at $4FFF, we now type $4FFF and a <RETURN>. Note that you always tell the BURN routine where in the address space of the 64 to look for the code to burn, and that you can burn just 1 byte at a time by giving the same address to FROM and TO prompts.

As soon as you hit that last <RETURN> the system erase checked the EPROM, to make sure that no zeroes are being asked to change to ones, which is forbidden. If the EPROM passes, the screen is now green, and the message SET SWITCH TO BURN is showing. If the screen is black and says ERASE EPROM, turn the mode switch to "O", change to another supposedly erased EPROM, turn mode back to C, and hit a <RETURN> to try again. Assuming you are being told by the screen to set the switch to BURN, flip the MODE switch to the B position, then type a <RETURN>

after you see the RED LED come on. The screen will be telling you what location is being burned. This burn will take 3 minutes, 31 seconds. During this time the 64 is not available for other purposes.

When the burn is done, the screen will turn blue and say "SET SWITCH TO COPY". Move the mode switch back to C and hit a <RETURN>. At this point the software checks the EPROM contents against 64 RAM. If any discrepancies exist, the screen remains blue and tells you the RAM address that is faulty on EPROM. Hit further <RETURNs> to see about more errors, or an X to end the burn check. To try burning another EPROM, change to a new one, place the MODE switch in "C" position, and type an E to erasecheck the new one, and proceed as before. If the screen says "BURN VERIFIED" and is green, type an X to get back to the Main Menu, and label your backup Autohex EPROM.

REMEMBER; NEVER BURN PAST THE END OF EPROMS. Since EPROMs shorter than 8k appear more than once on the PROMQUEEN 64 you could inadvertently burn over the beginning with errors.

3.3.2 Copying EPROMs to cassette recorder

Menu option C allows you to save to cassette directly from ROMs or anywhere else in C-64 memory. To do this, just type a C. Once again the FROM TO questions must be answered. Since the EPROM is at $8000 to $8FFF, you may be tempted to answer the TO prompt with 8FFF. This, unfortunately, misses the last byte. Be sure to add 1 to the "TO" address when saving to Tape or Disk. Give FROM $8000 to $9000. Next you will be asked for a device number, which is 1 for the cassette. Type a 1. Next you are asked for a name under which to save the code. You need a name for discs and it's convenient for tape. Make up a name (16 chars maximum) and hit a <RETURN>. You will be asked to press record and play on tape. After the save the display will re-appear, saying OK if all went well, or ERROR if it didn't. Press a <RETURN> to get back to the Main Menu.

If you saw OK, the save was good, and you have a copy on tape.

Note that Autohex saves code in a fashion that allows for relocation upon loading. The code will be loaded by the C-64 normal load routine to the start of BASIC, unless you specifically ask that it be reloaded where it came from with the command:

```
LOAD "PROG NAME",1,1
```

Also note that you have saved Autohex directly off its EPROM, not out of RAM at $7000 where the Autohex program is written to work. To get a tape version of Autohex that loads and runs properly with the above LOAD command, save it directly from $7000 to $8000, so that it is reloaded there by the C-64 operating system, if used, then the program can be run with the usual SYS 29181 <RETURN> command.

3.3.3 Copying EPROMs to RS-232

The Autohex program provides all the software you need to send and receive code over the RS-232 interface. All you need is a properly set up voltage adapter to convert the C-64 TTL levels to the ± potentials of RS-232, and an appropriately wired cable. See the instructions for your adapter to make sure you have the electrical set up for either DCD or DTD. The adapter setup must be the opposite of the setup for the other device on the line.

Sending code to RS-232 is the same as for tape, in that you must answer the prompts for where the code is to be sent. Then, when asked for a device #, type a 2.

Next you will be prompted for the control byte. This byte determines the baud rate, # of stop bits, and the number of bits per word. The control byte is explained in the C-64 Programmers Reference Guide, page 350. The default value of the 08 that comes up will give a baud rate of 1200, 1 stop bit, and 8 bit words. To use the default control byte, just hit <RETURN>. To change, type it over to what you want, then hit a <RETURN>.

Next the current command byte is displayed. The default byte of 00 defines no parity, full duplex, and 3 line handshaking. If this is OK, just hit a <RETURN>, otherwise consult page 351 of the C-64 Reference Guide for how to change this byte. Just type over and hit a <RETURN>.

As soon as you hit the <RETURN>, the channel will start to send, if the other device is ready to receive. The software logs each byte sent on the screen so that you know something is happening. When the last byte is sent, there is a pause while transmit buffer is emptied, then the Main Menu reappears.

You can discontinue transmission once started by

hitting the stop key. If errors occcur in transmission, the transmission is immediately terminated and the RS status byte at the time of error detection is printed to the screen. See page 355 of the Reference Guide to interpret the meaning of this byte. Hit a <RETURN> to get back to the Main Menu.

3.3.4 Saving Code to Printer

Aufcohex/64 provides a hex dump to printers which recognize device addresses 4 or 5 on the Commodore Serial Bus. Each line printed by the hex dump starts with the 4 digit hex address where the line begins, followed by a 16 hexadecimal numbers representing the contents of the start and next 15 addresses. The printout density is almost 1 kilobyte per page on Commodore printers.

The hex dump runs automatically whenever you SAVE to devices 4 or 5. If the printer isn't present at the specified device address, the message "IOERROR#5" appears, signifying device not present. (Kernal 10 error messages are explained in the Reference Guide on page 306.)

To print, just type a C to the Main Menu, give the address range in the 64 where the code to be dumped resides to the FROM/TO prompts, and type the number 4 or 5 (which address is your printer at?). If the printer is present, the dump begins immediately.

If you want a dump of, for instance, the C-64 BASIC interpreter, type a C to the Main Menu. Give the FROM prompt the reply: $AOOO <RETURN>. Give the TO prompt the reply $BFFF. Type a 4 if your printer is device #4, or a 5 if your printer is device #5. Be prepared to wait 20 or so if you have a 1515 printer. The resulting printout will be about 8 pages worth.

If you want to terminate printing, hold down the STOP key until the printer finally gets the message. Hit a <RETURN> to get back to the Menu.

3.3.5 Copying EPROMs to Disc *SEE IMPORTANT NOTE BELOW;

If the Autohex SAVE routine is given a device address from 6 to F, it assumes the device in question is a disk drive. The usual 1541 disk is addressed as device #8, but if you have additional drives at other device addresses, just give the number they recognize to the DEV# prompt. (You will have been asked FROM...TO... first after typing

a C to the Main Menu.)

Autohex always closes all files whenever the Menu appears. When it opens the file to a disk, it first gives the disk an initialize command to make sure the drive contains an up to date directory, so you are free to change disks at any time Autohex is not actually communicating to the drive. (Autohex also closes the command channel after each communication to make sure an up to date BAM is written back to the disk.)

So the SAVE to disk works the same as the SAVE to tape except for the device number and the fact that disk files must have a name to avoid error messages (names are limited to 16 characters or less).

All files saved to disk or cassette through Autohex are so specified as to allow them to be loaded to other than the original location in memory, if desired.

Save your copy of Autohex/64 onto disk. With the Menu displayed, type a C. Type $7000 to the FROM prompt (don't forget the <RETURN>) and $8000 to the TO prompt. Put the disc you want to it on in your drive, and type the applicable device # (usually 8).

When you want to reload the program from disk to the C-64 when in the normal BASIC operating system, be sure to use the command

```
LOAD "AUTOHEX 64",8,1 <RETURN>
```

Giving the secondary address of 1 tells the Kernal to put the code back where it came from in memory.

Once the program is loaded, type SYS 29181 <RETURN> to run it.

IMPORTANT: ALWAYS SAVE to Tape or Disk ONE more byte than you would otherwise. The KERNAL save routine saves up to, but not including, the TO address you give to the Autohex prompts.

3.4 Loading code to RAM

You have available 5 different ways to get code into C-64 RAM so that it can be burned on EPROM. You can:

- Type it in from the keyboard
- Copy it from compatible ROM and EPROM devices

- Load it over an RS-232 line
- Load it from tape
- Load it from disk

Loading of code from EPROMs and ROMs is done with the T function of the EDIT HEX routine. Keyboard entry is also performed by EDIT HEX, and can also be done with other monitors and/or assemblers. Use these just as you usually would without the PROMQUEEN 64.

The tools in EDIT HEX are explained in detail in the EDIT HEX Tutorial, so they will be deferred from the present discussion.

3.4.1 Loading from RS-232

To load from RS-232, select Menu option D. Type a 2 too the DEV# prompt. The current control byte then appears. Edit it if required and press <RETURN>. Just press <RETURN> if it is OK. Next the current command byte appears. Edit and/or press <RETURN> (control and command work the same when receiving RS-232 as they do when sending it). Next the usual FROM...TO... prompts will ask you where to put the code received. When you hit <RETURN> after the TO address, the cursor will disappear and the system will devote itself to looking for input. At this point you should start the sender sending. As soon as the C-64 starts receiving from the RS-232, it will confirm the receipt of each byte by displaying the message "RECD" and the address where the byte was put on the C-64 screen. This lets you know that something is happening.

If you want to terminate the load, hit the stop key. If any errors are detected, the load will terminate and the status byte at the time of the error detection will be displayed. If the load goes to completion, the Menu is displayed. It's a good idea to tell the sending system to send a few extra bytes to be sure and drive the C-64 to completion. If you forget to send enough, the stop key will free you up. Hitting a <RETURN> after error messages gets you back to the Menu.

3.4.2 Loading from cassette

Type a D from the Menu, and a 1 to specify cassette. Now you will be asked "NEW LOC?". You can put the code where you want by typing a Y, PROVIDED that the original SAVE allows relocation. (All saves through Autohex allow relocation.)

If you type a Y, you will be asked "WHERE?". Give the address where you want the code to start. (Don't forget the $ sign if your entry is in HEX, or a # sign if your entry is in decimal, and hit a <RETURN> to conclude your entry.)

If you type anything other than a Y, it will be interpreted as a NO, and you won't be asked WHERE? The file will load where its header says it was saved from. Files, which were saved without permitting relocation, won't relocate even if you say Y and give a load address.

Next you will be asked for a NAME. If you want a specific file to load, give its name. No name will load the next file found on the tape.

When you hit a <RETURN> after the NAME, the system will prompt you to press play on the recorder. When the cassette finds the program, press the space bar to continue with the load. When the display comes back it will tell you OK if the load was good, or ERROR if anything went wrong. Hit a <RETURN> to get back to the Menu. To terminate during the load, just hit the stop key.

3.4.3 loading from Disc

This works the same as for cassette, except you have to give a valid disc device number, and all files must be named. Just put in the disc you want to load from, and type a D to the Main Menu. You may have the option to relocate disc files in memory the same as for tape. If the load goes well, you will be told OK. Press <RETURN> to get the Menu. If there is an error, it will say so and pressing <RETURN> will get you to the Menu.

3.5 Verifications

Autohex 64 allows you to verify that files you have saved to tape or disk are in fact a copy of what remains in memory after such saves. This utility is accessed through the Main Menu option "E" V'FY SAVE. In addition the BURN EPROM option "B" on the Menu allows you to check EPROMs against a block of code serving as a reference standard which you have previously put in a designated block of C-64 RAM. You can also check EPROMs for erasure, that is that all bit locations are set to "I".

3.5.1 Verifying saves to tape or disk

If you have just saved something to cassette (device #1) or disk (device addresses possible are 6 to F) , you

can verify the save as long as the code still remains in the C-64 where it was saved from. Just type an "E" to the Main Menu. You will be asked which device number to verify against. Next you will be asked the NAME of the file. Give it, then hit <RETURN>. If the file is on disk, the verification will begin immediately. You will get the usual control messages if the device is the cassette. The screen blanks during the verification. When the verification is done, the screen display returns. The last line of the display will say "ERROR" if any mismatch was found, OK otherwise. Type a <RETURN> to get back to the Main Menu.

3.5.2 Verifying burned EPROMs

When you are burning EPROMs the Autohex program automatically erasechecks the EPROM first and verifies the burn afterwards. You can also check many EPROMs against an internal reference standard you have put in C-64 RAM without having to re-enter data. Let us try verifying the Autohex EPROM.

Any reference must first be copied into C-64 RAM. Put your Autohex EPROM on the ZIP socket (make sure the MATRIX is still set for 2732) and turn on the 64 to load Autohex.

Next we must use the Autohex to make a copy of itself. We use the TRANSFER function in the EDIT HEX routines for this. Type an A to the Menu to get the EDIT HEX package. We shall choose $2000 to $2FFF as the C-64 RAM space where we will put our reference, so type $2000 after the WHERE prompt (and hit a <RETURN>). When you see the display at $2000, hit the "T" key to call the transfer, and answer $8000 to the "XFER FRM" prompt (hit a <RETURN>. When the STOP@ prompt appears, make sure the four digit hex number that follows is 2FFF, and hit <RETURN>. You should now see the start of our reference code appear on the screen. Type an X to get back to the Main Menu, our transfer is now complete.

Now type a B to go into the BURN EPROM routines. Type $2000 after the "FROM" prompt. Hit a <RETURN>. Type $2FFF after the "TO" prompt, hit a <RETURN>. If we left our Autohex EPROM on the active ZIF socket, the green screen will now say SET SWITCH TO BURN. This is because the Autohex erasecheck asks only that the proposed burn not require that any "0"'s be changed to "l"'s on the EPROM. If there are any 0's to be changed to 1, the screen will come black and tell you to "ERASE

EPROM". If there are zeroes on the EPROM only where zeroes are to be burned, you will be allowed to proceed.

Either way you can proceed with verification by typing a V. When you do this, the screen will turn blue, and tell you to SET SWITCH TO COPY. Check that the mode switch is in copy position, then hit a <RETURN>. The screen will blank momentarily, and the message "BURN VERIFIED" will appear if the verification goes to completion. Otherwise, the address at the location in C-64 RAM found to differ from the EPROM contents will be displayed. Hitting further returns will disclose further errors, if any, or you can type an X to force the verification to completion.

Once the verification is complete, you can change EPROMs to verify another. Turn the MODE switch to "O", swap on another 2732 type EPROM, and then type a V. You will be reminded to set the MODE switch to COPY position. Do so. Now type a <RETURN> to initiate the verify again. You can do this as many times as you want to.

Verifications of this type take about 1/2 second for 4k EPROMs, or about 1 second for 8k EPROMs.

3.5.3 Verifying EPROM erasure

EPROMs are erased when all bit positions are set to 1. You can check a batch of EPROMs quickly to find out if they are erased.

You must always set a reference RAM section in the C-64 to all "l"s to perform a true erasecheck (remember, the software intentionally allows you to burn unerased EPROMs if the burn can come out correct).

To set a block of memory to all "l"s you must use the STUFF function in the EDIT HEX routines. Type an "A" to the Main Menu, answer the WHERE prompt with $2000 <RETURN>, and type an S to call the STUFF function. The screen should now be saying (on the lower left) "STUFF TO 2FFF". The STUFF function uses the "STOP@" number to define its upper limit. If it doesn't say STUFF TO 2FFF, type over until it does, otherwise just hit a <RETURN>. Then you will be prompted by the message "WITH" and a blinking cursor. Type two "F"s and hit a <RETURN>. You will see all FFs appear from $2000 up on the display. Type an X to get back to the Menu.

Now type a "B" to get into BURN EPROM mode. Answer

the FROM with $2000 <RETURN> and the TO with $2FFF <RETURN>. If you left your Autohex EPROM on the socket, the screen will be black, and you will be asked to ERASE EPROM. Change off the Autohex EPROM (remember to use "0" MODE when changing EPROMs) to a blank 2732, and put the cartridge back in C mode with the toggle switch on the cartridge.

Now hit an "E" to erasecheck the new EPROM. The screen should blank momentarily, then come put green saying "SET SWITCH TO BURN". This indicates an erased EPROM. If the screen is black, the EPROM needs erasing.

Now you can repeatedly check EPROMs for erasure. Just change EPROMs and type an E after you've set the cartridge mode back to "C". The screen always blanks momentarily, but comes up green on erased EPROMs, black on ones that need erasing.

When done checking a batch, just type an X to get back to the Main Menu.

3.6 Editing Hexadecimal Files in RAM

3.6.1 Introduction to EDIT HEX

So far we have discussed getting code into and out of RAM from devices and EPROMs. Now we shall pass on to entering and editing it from the keyboard.

Autohex 64 has a package of tools for doing this, all collected under the EDIT HEX option A from the Main Menu. As soon as you type the A from the Menu, you are always asked "WHERE?". You must decide what location in memory at which you want to work. You can answer this question three different ways, in hex, in decimal, or by MARKER, in a fashion to be covered in detail shortly.

Once you have told the program where in memory you want to work a "window" into memory opens, showing the contents of the address in question, as well as a range of the surrounding addresses and contents. When this display is present, the following keys on the keyboard become active for selection of options and entering data direct through the window into RAM;

- All number keys plus hex digits A-F
- Keystroke commands I-T inclusive
- All of the grey function keys
- The cursor up and cursor down keys

- The X (abort) key.

A tutorial on the use of these functions follows the below list of available keystrokes.

AUTOHEX/64 KEYSTROKE COMMANDS IN EDIT HEX MODE

I INITIATE

is used to start the machine code routine beginning at the byte located under the blinking cursor in the display.

J JUMPBACK

to display the address which had been displayed before you used the *L, P,* or f8 keys.

K CONVERT

single and two-byte decimal numbers to Hex and enter them. 2-byte numbers are entered low-order byte first.

L LOOK

at the destination of an offset under the blinking cursor. Jumpback by typing J to return.

M ASSIGN A MARKER

numbered from 0 to F in Hex to the address at the blinking cursor.

N LOAD THE ADDRESS

at which the designated marker was placed to the location of the blinking cursor, low-order byte first.

0 FIND OFFSETS

to and from markers, and enter at either end of the jump, one end defined by the blinking cursor, the other by the Marker.

P POSITION

the display at the location whose low-order byte is presently displayed under the blinking cursor. Use J to return.

Q SEARCH
for a specified 16-bit operand. Put the display there.

R REMAP

by adding a specified offset to the 16-bit operand whose low order byte appears under the cursor.

S STUFF

a range of bytes from the blinking cursor up to the designated stop address with a specified byte.

T TRANSFER

a block of code starting at a designated address to the range of addresses which begin at the blinking cursor and end at the designated stop address.

THE HEXKIT FUNCTION KEYS

f1 C/MODE on/off toggle

f2 Insert up to stop address

f3 If in C/MODE, insert a zero stop byte

f4 Delete up to stop address

f5 Scroll display to next lower address and bip

f6 Alternate set Marker keystroke that works in C/MODE

f7 Scroll display to next higher address and bip

f8 Reposition display to new address. Jumpback with J.

X Your general purpose abort key

CURSOR UP - CURSOR DOWN

Repeating display scroll

3.6.2 Autohex 64 tutorial : selecting WHERE to edit

As mentioned earlier, you are always asked, whenever you type an "A" to the Main Menu, WHERE in C-64 memory you want to edit. You can go anywhere you want, including the display generator chip, SID chip, etc. if you want to see what is there and/or try changes. But if you want to use RAM workspace to develop code, you must avoid conflict with the built in functions of the C-64 as well as whatever other support software you might be using. Whenever you are using the PROMQUEEN 64 with the Autohex software alone, the entire range of memory from $800 (decimal 2048) to $6FFF (decimal 28671) is open for your use. Of course once you start using sprites and/or Hi-Res mode, some of this will disappear. Be sure to understand how to use memory for these C-64 features when you use them.

Other than avoiding conflicts with software loaded in the C-64, the only other thing to think about is to position the code you want to write in C-64 memory so that it maps properly onto the EPROM when burned. This means that your code should start on an even 8k boundary if it is to be burned on an 8k EPROM, any 4k boundary if it is to be burned on a 4k EPROM, any 2k boundary if it is to be burned on a 2k EPROM, etc.

To be safe on any size EPROM up to 8k, you can consider using 2 different workspaces - $2000 (8192 decimal) to $3FFF (16383 decimal), and $4000 (16384 decimal) to $5FFF (24575 decimal). If you are developing 16k worth of code to burn on a 27128, use from $2000 to $4FFF.

Fire up your PROMQUEEN 64 with Autohex to get the Autohex running on the C-64. Turn up the sound on your monitor, Autohex/64 has auditory feedback to help you avoid mistakes. Type an "A". When it comes up saying "WHERE?" type a $ sign which tells the program that you are answering the question in Hex, then the hex digits 2000. Hit a <RETURN> and see the display. Notice that you see 3 columns of information. The column of 5 digit numbers on the left is the address of the displayed location numbered in decimal.. Next comes the representation of the same address number in Hex (4 digit). The two digit number is the 1 byte number in the address, displayed in Hex. There are also two additional columns for information. The leftmost column is used for MARKERS numbered from 0 to F that you can tag onto an address and use for address reference, calculating

offsets used in relative addressing and loading jump tables, subroutine calls, etc. You can mark the current displayed location by typing an M, and then any number you choose between 0-F, so up to 16 MARKERS may be maintained at once. Mark the present location as M0 by typing an M, then a zero. Now type an X to take you back to the Main Menu. (X takes you back to Menu out of the normal EDIT HEX display.) Type an A again, and this time answer "WHERE?" by typing an M and a 0. Notice you now see the display where you assigned the marker.

You can also answer "WHERE?" in decimal. Just precede your decimal entry with a # so the program knows you want decimal entry. If you type an X, then an A, then #8192 <RETURN>, you'll be back at the same point as before. Don't bother to type more digits than you need to specify the number, the software accepts input properly without leading zeroes.

Once you have a display at one location in memory, you can move the display to any other location by typing SHIFT F7 (special function key F8) . The prompt DISPLAY and the cursor will appear at the lower left of the screen. You can give an address in decimal, hex, or by MARKER exactly as you did before to the WHERE prompt. Try looking at address $7000. Hit SHIFT F7, then $7000 <RETURN>. See the display is now at $7000. Mark this location as MF by typing an M, then an F.

When you relocate the display this way, it is frequently handy to be able to pop back to where it was before. Autohex/64 provides the J keystroke for this purpose. Type a J now and see the display pop back to $2000. This J for jumpback works after the diagnostic display relocations (discussed later) as well.

If you try the DISPLAY function again (SHIFT F7) and this time type an M and an F, you will see it work on MARKERS. Decimal entries work if proceeded by a # and followed by a <RETURN>.

Get back to address $2000 by use of J, or SHIFT F7 M0. Try tapping the F7 key. Notice the display scroll down one address for each tap, with a lower pitched bip. Hold down the cursor up key and watch the display scroll continuously and silently towards higher addresses. Hold down the cursor down key and observe the display scroll continuously towards lower addresses.

Hence you can specify where you want to be in memory

in *hex,* decimal, or by MARKER, use the J for jumpback key to get back to the previous displayed location, and scroll around locally in discrete steps or continuously. By the way, if you call any such function, type an X to abort if you need to. If you have already typed #, $, type 2 X's, and <RETURN> to abort.

3.6.3 Typing in Hexcode

Before proceeding with the tutorial, let us briefly review the basic concepts of machine code programming.

Any processing unit appears to the programmer as a set of registers, a set of instructions which the processor can execute, and a set of addressing modes by which the processor can be told how to find the data on which it is to operate.

As far as the programmer is concerned, the 6510 processor in the C-64 has an X register, a Y register, a status register, a stack pointer, an accumulator, and, the only 16 bit register, the program counter. There are also a variety of addressing modes, not all of which are relevant to each instruction. There are a grand total of 151 instructions, many of which perform similar functions, but with different addressing modes. Each of the 151 instructions has its own opcode, which is an 8 bit number, and thus fits in one byte. If the processor is fetching a byte from memory and expecting an opcode, the number it gets tells it what to do next. Appendix I lists the entire set of 151 opcodes in the hexadecimal form that you can just type right in with Autohex/64.

6510 instructions can be 1, 2, or 3 bytes long, depending on the addressing mode. The additional 1 or 2 bytes following the opcode provide whatever additional information, if any, is required by the particular addressing mode of the opcode in question. Many instructions, such as PHA (push accumulator contents onto stack) or TAX (transfer accumulator contents to X register) imply where the operands are to be found, so these instructions are 1 byte only, just the opcode. Other instructions may use the immediate addressing mode (the operand is the byte following the opcode), or a zero page mode (the operand or a pointer to the operand is in the first 256 bytes, or zero page, of memory). Such instructions are two bytes long. A third set of instructions refer to addresses anywhere in memory, which takes two bytes to specify, so these instructions total 3 bytes in length. Each of these bytes is just an 8-bit

number, which you can edit directly through Autohex/64.

Let us type a machine code program into the C-64. The program will print the C-64 characters from down to ! to the screen. Let us put the program at $2000. Use your ability to position the screen to put it there.

The first thing we want to do is clear the screen. This is done by loading the accumulator with ASC (SHIFT CLR HOME), as it would be expressed in BASIC, then calling a Kernal routine that prints to the current output channel (right now thats the screen). The code for LDA (load accumulator) immediate (with the next byte of the program) is A9. Type an A and a 9 and see it appear at $2000. The screen scrolls up 1 location.

REMEMBER If you should type an incorrect Hex digit on the first nybble of the code, type an X to remove it. If the second nybble is at error, back up the display one position and type your entry over.

Now we need to give the SHIFT CLR HOME byte. Commodore's manuals tell us this is 147 in decimal. Autohex/64 will convert this to Hex and enter it for us. Just type a K. The prompt CNVRT DEC# appears on the bottom left of the screen. Type in 147 and hit <RETURN>. You will see the hex number 93 appear in the code at $2001.

All Kernal routines are structured as subroutines, so we need to call the Kernal CHROUT routine by using a 20 opcode for jump subroutine, then 2 bytes that specify the address of the subroutine, low order byte first, CHROUT is at $FFD2, so we now type a 20, then a D2, then an FF, which appears on the display as we type.

Now we need to set up an index by which to count. Let us run the count in the X register. We must load the X register with the ASC for , since we will be counting down. To do this in the immediate addressing mode we type an A2, then call the convert (K) keystroke again, and convert 126 in decimal (the ASC for) and autoload it after the A2. The number 7E will appear on the display after you type <RETURN> after the 126.

REMEMBER If you type an incorrect function key, try typing an X. If that doesn't abort it, type another X and hit <RETURN>. That will abort it for sure.

We will be running a loop to print a number of

characters. The next byte to be entered is in the start of the loop. Let us MARK this location as MO so Autohex can calcculate the necessary offset for us later. Type an M, then a zero.

The CHROUT routine we are using to print out to the screen expects the ASC of the character in the accumulator. We must transfer the contents of our counting register (the X register) before we can call this routine. So let us put a TXA command (implied addressing - only one byte) in this location we just marked by typing an 8 then an A.

Now we can jump subroutine to FFD2 again (the CHROUT KERNAL routine) by typing a *20,* then a D2, then an FF. See this entered on the display.

Next we must decrement the count we are running in the X register. This is done with the one byte DEX command, the opcode for which is CA. Type in the CA.

Next we must find out if X has counted down past the ASC value of the last character we want to print. We must compare the X register with our lower limit. For immediate addressing of this X comparison we use the opcode EO. Enter it. We want to print the ! (ASC value of decimal 33), so our lower limit has to be decimal 32. Use the K (CONVRT DEC#) function again to convert and enter decimal 32 to the code. You should see a 20 appear on the display.

We want to branch back to the beginning of our loop to print the next character as long as X has not yet counted down to our limit. We need a BNE (branch if not equal) command to do this. This opcode is DO. Type it in. All 65XX branch instructions use what is called relative addressing, and total 2 bytes in length, including the opcode. The second byte gives a signed 8 bit number which is to be added to the contents of the program counter if the branch condition is met to point to the next instruction to be executed. Because this single byte offset is signed, you can't go back more than 128 addresses, or forward more than 127 addresses when using relative addressing.

Autohex/64 will calculate the right offset for us because we marked the destination as MO. To have Autohex do this for us, we just type an 0 for offset. The prompt LD OFFSET M and the cursor appear as usual on the lower left of the screen. Type a 0 to tell if the marker in

question is MO. Next the prompt "PUT H or T". Autohex allows offsets to be loaded at either end of loops. In this case we want the offset right where the display is, so we type an H to specify HERE. Other situations may require you to put the offset back where the marker was assigned, if the offset is a forward jump. The appropriate offset (F7) is entered as soon as you type the H.

Now we are at the end of our machine code program, so all we need to do is enter an RTS (return from subroutine) opcode. This is just a 1 byte instruction, and the opcode is 60 in Hex, so type in the 60.

Let us just check our program, go back to $2000 (SHIFT F7 then $2000 <RETURN>) and check to see that what you see is the following list of code as you scroll up in memory from $2000:

A9 93 20 D2 FF A2 7E 8A 20 D2 FF CA EO 20 DO F7 60

The last byte of the above should be in address $2010. If you find any discrepancy, correct it now.

Autohex/64 lets you run code programs in C-64 RAM on the 64, so you can try them out conveniently. All you have to do is position the display on the entry byte of the (in this case $2000), then type an I for initiate keystroke. Position the display so that $2000 is the line where the cursor is blinking, then type an I. The character set will be displayed on the top of the screen.

The Initiate keystroke does not bring you fully back to Autohex when used, so that you can see the effect of the code you have tried. Just type a <RETURN> to return fully to Autohex.

See how easy it is to write code?

Autohex/64 still has many tools that we have not used yet. There are, for instance, two diagnostic keystrokes, L for looking at offsets, and P for pointer display. If you use these to trace your code, you will avoid the major cause of machine code crashes - addressing errors.

Try out the L for lookat keystroke on the offset you had the Autohex calculate for you above. Scroll the display so that the blinking cursor appears on the F7 byte at $200F. Type an L. See the display now located

at 2007. This location should still be marked as MO, and contain an 8A byte, telling the 6510 to transfer X register contents to accumulator. This is the location at which program execution will continue if the BRANCH IF NOT EQUAL (DO) instruction in 200E results in a branch. this is correct now. Let us assume that this was a mistake - that we really wanted to have this offset take us back to $2000. To correct such a mistake we just move the marker to the right location. Just scroll back to $2000, type an M and a 0, and see the marker reassigned. Now type a J for jumpback. You'll see yourself back at $200F, which currently holds the byte F7. Recalculate the new offset with the 0 keystroke the same as you did before, and check with the L keystroke that the new offset does go to $2000.

The P keystroke performs the same way as the L keystroke except that it works on absolute address calls rather than offsets. Scroll the display to address $2003. This address contains the first (low order) byte of the address at which call the KERNAL CHROUT routine. We can begin to see how the KERNAL works by now typing a P. Notice the display has relocated to address $FFD2. This location contains an opcode, 6C. This byte tells the 6510 to jump to an address whose value is contained in the address given as the next two bytes (as always, low order byte first) . We can continue to trace along with the execution of the program by scrolling to the next byte (it's the low order byte of an absolute address call) and typing another P. The display now relocates to $326. This address contains the vector to where execution is supposed to continue, so typing another P takes us to the continuation of the program. Do it. Now we see the body of code that deals with putting a character on the screen. The 48 means Push accumulator contents onto stack. The next two bytes tell the 6510 to load its accumulator with the contents of address $9A (look in your reference guide to see that this location contains the current output device number) (on page zero). The next two bytes compare accumulator contents to 3. If the comparison reveals enequality, the program branches 4 steps forward, otherwise it continues...

The J for jumpback takes you back to the last screen position before use of P for point, the same as for the L for lookat keystroke. But remember, J only remembers the last address previously displayed, successive use of the P keystroke does not load up a return stack for multiple use of the J keystroke.

There is still much more to Autohex/64, so let us work up another example to exercise more tools in the program.

First we should review the MODE switch on the cartridge. Remember that the cartridge in "0" MODE is removed entirely from C-64 memory. Hence you can use the RAM in the 64 from $8000 to $9FFF. Since this is the same address space as occupied by ROM cartridges, we can write and debug code here and burn it on an EPROM without having to remap (change it so it works at a different address) the code.

Fire up your Hexkit (perhaps it is still up) , turn the MODE switch to "O", and position the display at $8000. You are looking at C-64 RAM now, not the EPROM on the cartridge.

You can use Autohex to clear out a section of RAM to any byte value desired. This is the STUFF function, called by typing an S. Type the S and see the prompt STUFF TO followed by a four digit Hex number, which is the last STOP@ address you used. Type over this number with 9FFF and hit <RETURN>. Then the prompt WITH: will appear. Let us set this 8k block of RAM to FF. Type FF and hit <RETURN>. The program will take about 1 second to perform this stuff. When the normal display reappears, you can move around in this block and see that all bytes have been set to FF.

Autohex/64 has a Transfer function that allows you to pick out blocks of code from elsewhere in C-64 memory, and put where you want to. This routine is bidirectional so you don't need to worry about the accuracy of overlapping transfers. Let us use this on another Autohex routine.

Autohex/64 has a routine in it which is used to print out the various messages and prompts you see when using the program. This routine is called with the number code for the desired message in the accumulator. The routine uses this number to look up from a table the address in memory where the desired character string resides, then it prints the string to the screen. The end of each string is denoted with a zero byte, so the routine returns when it reads a zero.

Let us transfer up this routine out of where it resides in Autohex when Autohex is loaded in the C-64. The transfer function is called when in EDIT HEX mode by

typing a T. It is designed to transfer code from a base address given by you to the new base at the address on the display where the cursor has been blinking. Thus you must always have the display positioned at the address which is the new base address of the block of code you want to transfer, before you type the T. When the function operates it copies as many bytes as it takes to fill the space between the blinking cursor location and the location given as the STOP@ address that defines the end of your workspace. If the STOP@ is a lower address than the cursor address, the transfer will abort.

So be sure the display is at $8000, and that the cartridge MODE switch is set to "O". Type a T. Notice the prompt XFER FRM. Answer this with $7B27. (Remember, you can also answer this with a MARKER #, or in decimal.) Hit a <RETURN>. Then type over the STOP@ address to 8100. This will copy 257 bytes, more than enough. Hit a <RETURN>. Notice the code appear starting where the blinking cursor had been.

We will want to reference the start of this routine later, so mark it as MO. (Use the M keystroke.)

We deliberately copied much more code than required to illustrate another Autohex/64 keystroke, Q for SEARCH. This SEARCH looks for two bytes in sequence that match the address call or other (possibly reversed) two byte sequence, and positions the display there, ready for you to edit any revisions you may want to make. If you want to, for instance, find a call to the KERNAL routine CHROUT, tell the search you are looking for FFD2. The routine we just transferred contains such a call, let us find it. Type a Q. Type over whatever four digits appear following the SEARCH prompt with FFD2. Hit a <RETURN>. See the display positioned on a D2 byte, the low order byte of the address portion of the instruction JUMP subroutine at $PFD2. If this routine had been moved, we could change this call address now.

We can also use the SEARCH to find a particular location in our code if we can give it two bytes at that position in REVERSED order (REMEMBER - this search looks for address calls, which are always low order byte first in 65XX processors) . Thus if we know that this routine we just copied ends with the two byte sequence EF60 (it does), then we find the end by searching for 60EF because we have to reverse the byte order. Type a Q, change that FFD2 to 60EF (notice how it remembers the last thing you were searching for), and hit a <RETURN>. See the display

move to $801D. The 60 in $801E is the last byte, so we need to scroll down to $801F to be past this routine. Do so.

Aufcohex/64 allows us to open and close spaces in the code with INSERT and DELETE functions. Now is a good time to try this. Type a SHIFT Fl (special function key). See the prompt INSERT and the STOP@ address of $8100 of before appear. The STOP@, as usual, defines how far up in memory the operation will occur. As usual with any remembered number, you can abort by Xing out the number and hitting a <RETURN>, or change it and hit <RETURN> to update it to some new value. In this case, the insertion will not push up anything beyond $8100, and there's no reason too change this, so hit a <RETURN> to re-use it. See the code move up in memory, starting at the location where the cursor has been blinking.

The DELETE, which works with the STOP@ value in the same way as INSERT, is called by typing a SHIFT F3. Try it. Hit <RETURN> to re-use the current STOP@ address. See the code shifted down in memory towards the location where the cursor has been blinking.

You can make multiple insertions and deletions with only two keystrokes each, because the program remembers the STOP@ address you set to define the upper end of your workspace.

With your cursor on $801F, use the STUFF function (keystroke "S") to clear out memory to FF. The STOP@ 8100 need not be changed. Just hit a <RETURN> to re-use it.

Remember that the routine we just transferred requires that the accumulator must be preloaded with a number that is used as an "index" to look up the address of the string to be printed. Let us reserve some space for this table, starting at $801F. Mark this location Ml, we'll need to reference it later. If we want to leave room for 16 message "vectors", we need 32 bytes, so let us move the display to $8040.

Autohex/64 has a special entry mode for character strings, called C/MODE. The program toggles into or out of C/MODE every time you type an Fl special function key. When in C/MODE the cursor is blinking in the 5th column of information on the screen, and it says "C/MODE" in the prompts window. Try hitting Fl a few times, see how it toggles.

Perhaps you have seen characters scrolling by in that 5th column of the display. The purpose of this column is to display any alphanumeric characters in VASCI represented by the hexcode in the address of the line. When in C/MODE you type the characters on the keyboard directly to enter the appropriate code, and see alphanumeric characters displayed as you enter them. This works for all keys except the special function keys, which remain as control keys. If you type characters for color change, cursor movement, spaces, etc., as well as graphic characters, these too are automatically entered, even if they don't appear in the 5th column.

It's handy to MARK the beginning of character strings when in C/MODE, so you can later load the access table conveniently. Toggle into C/MODE and use SHIFT F5 to mark location $8040 as MA, by typing an A after the M prompt. If you do any scrolling in C/MODE, be sure to do it with F5 and F7 provided for this purpose, not the cursor keys, which will instead get entered in Hex to the code. If you have to relocate the display, insert, or delete when in C/MODE you are able to do so normally.

Let us type in a character string at $8040. In C/MODE type the following:

(control red) THIS PROMQUEEN 64 CARTRIDGE

See it appear as you type. It should end with $8055 as the next entry point.

The message routine detects end of message by reading a zero byte. Autohex lets you insert such a byte by typing an F3 special function key. Try it. This works only in C/MODE to avoid the confusion that might result if it were used inadvertently in normal mode.

Now we should be at $8056. Mark this as MB with the SHIFT F5 keystroke and a B. Type in the string:

(control blue)(space) IS A VERY POWERFUL TOOL

and end it with an F3 to put that critical zero stop byte. Now you should be at $8070. Mark this as MC (SHIFT F5), and enter:

(control black)(space) ISNT IT?

Don't forget the stop byte. You should be in $807B when

done. Toggle back out of C/MODE by tapping Fl.

Remember the vector table that accesses the strings? Let us load it now. We had marked this location as Ml so we can get there by using SHIFT F7, then an M and a 1. Do so.

Since we labeled the starts of our strings MA, MB, MC, we can load the vector table with the "N" keystroke, which loads to the address where the cursor blinks two bytes which are the address in the C-64 where the marker was assigned, low order byte first. Just type an N, then an A. See the vector entered. Now type an N again, then a B. See another vector entered. Repeat this again for MC to enter the third vector. You can test that these vectors point properly with the P and J keystrokes as before.

We have to do something to the original message routine in order to make it access the vectors to the strings we just entered in the table. The original routine is written to look at another table elsewhere in memory. In other words, we must "re-map" the routine so it works properly. Go back to the start of it by typing a SHIFT F7 then an M and a 0. We had marked it as MO so we can do this.

The original table base address in this routine is $7F24, so we can find the instances when the routine is looking at the table by using the "Q" search set to 7F24. After you hit a <RETURN> the display should be at $8003. This pointer, part of a "load the accumulator from an absolute address indexed by X register contents" instruction, needs to point at our new table.

For doing this repeatedly Autohex/64 provides an R for remap keystroke that remembers an "offset" to add to such pointers to point them to a new location. You figure out the offset to use in one of two ways, depending on whether the new pointer is lower in memory than the original, or higher.

If the new pointer is lower in memory then

offset=$10000+desired pointer-original pointer

If the new pointer is higher in memory then

offset=desired pointer-original pointer

You can do this calculation in decimal and use the K keystroke to find the hex conversion, but be sure to go somewhere in memory which won't be damaged when K enters its result.

In the present case the original pointer is $7F24 and the new pointer desired is $801F so the offset needed is

```
  $801F
- 7F24
  $OOFB
```

Once we know the offset needed in Hex, we just type an R (be sure the display is at $8003 as found by the Q search). The window then says "REMAP:ADD" followed by four digits of Hex. Type these over with OOFB. Hit a <RETURN>. See that the pointer has been changed to 801F, entered low order byte first, and the display scrolled two positions up in memory.

Now use Q again to find the other table lookup in the routine. It remembers you are looking for 7F24, so just hit a <RETURN> to execute. Then use R again, once again it conveniently remembers, so just type <RETURN> to execute. That's it.

Things have now developed to the point where we can test the setup. If what we have done so far works, we should be able to print any one of the 3 strings by loading the accumulator with 0, 1, or 2, then jumping to the message routine at $8000. Let's try it with a brief test routine at $8100. Put the display there with the SHIFT F7 function.

We can load the accumulator with the byte following the opcode by using the opcode A9. Type the A9 at $8100. To print the zeroeth message, we need 00, so type it in after the A9. We jump to the message routine with a 4C, so type that in. We can use the N keystroke to load the address of MO , because MO marks the start of the routine.

Test the code by backing up the display onto $8100 and hitting an I for initiate keystroke. See that (if you haven't made any mistakes) the first string shows up in the prompts window in red. Hit a <RETURN> to re-enter EDIT HEX. Change the 00 in $8101 to 01, back up to $8100, and hit an I again. See that you get the second string. Hit a <RETURN> back to EDIT, and change the 01 to 02, backup, and initiate at $8100 again. See the third string? Type a <RETURN> to get back to EDIT HEX.

Let us close with one more routine that will print all 3 messages in order. We can put this one at $8100 also. Get the display there by any appropriate means available.

We need to run a counter to print the messages in sequence, but the message printer routine unfortunately uses both the X and Y registers, which would foul up a count there. So we will run the count on the top of the stack instead. Enter the following routine at $8100:

```
          $8100    LDA#$00       A900
MF        8102     PHA           48
          8103     JSR MESSAGE   20 00 80
          8106     PLA           68
          8107     CLC           18
          8108     ADC#$01       69 01
          810A     CMP#$03       C9 03
          810C     BNE MF        DO F4
          810E     RTS           60
```

Back up to $8100 and hit an I. All 3 strings should appear in the prompts window. Type a <RETURN> to get back to EDIT HEX.

If we had just written and debugged this code for use in the same address space where it presently resides, as for a ROM cartridge, and we want to burn it on an EPROM, we must always remember that the PROMQUEEN 64 when in C or B mode occupies this space from $8000-$9FFF. We must move the code out of the way of the cartridge in order to be able to burn it. We can't verify the burn unless we do this. This is another use of the T for transfer keystroke. We can copy the code to any available even 8k boundary, such as $2000, $4000, or $6000 so that it will map properly onto any EPROM when burned. If the EPROMs we're burning are shorter than 8k, we'll have more options where to copy the code to.

You can use Autohex to construct auto starting ROMs as well. Appendix II will give you the proceedures, both for machine coded programs and for BASIC ones.

To end use of EDIT HEX, just type an X to get back to the Main Menu. Type another X to get back to VIC BASIC. Remember you can restart Autohex 64 when loaded with SYS 29181 <RETURN>.

If you have worked through successfully to this point, you have earned your diploma in Autohex/64. We hope you find it a convenient and powerful tool.

APPENDICES

APPENDIX I : **65XX** OPCODES IN HEX

BPL	10	BMI	30
BVC	50	BVS	70
BCC	90	BCS	BO
BNE	DO	BEQ	FO

Branches –0

ABS	(IND)	
JSR	20	
JMP	4C	6C

Jumps

Single-Byte Op Codes -0, -8, -A

	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	BRK				RTI		RTS									
-8	PHP	CLC	PLP	SEC	PHA	CLI	PLA	SEI	DEY	TYA	TAY	CLV	INY	CLD	INX	SED
-A	ASL-A		ROL-A		LSR-A		ROR-A		TXA	TXS	TAX	TSX	DEX		NOP	

	IMM2	ZPAG2	Z,Y 2	Z,Y 2	ABS 3	A,X 3	A,Y 3
ASL		06	16		CE	IE	
RCL		26	36		2E	3E	
LSR		46	56		4E	5E	
ROR		66	76		6E	7E	
STX		86		96	8E		
LDX	A2	A6		B6	AE		BE
DEC		06	D6		CE	DE	
INC		E6	F6		EE	FE	

Op Code Ends in -2, -6, or -E

	IMM	ZPAG	Z,X	ABS	A,X
	2	2	2	3	3
BIT		24		2C	
STY		84	94	8C	
LDY	A0	A4	B4	AC	BC
CPY	C0	C4		OC	
CPX	E0	E4		EC	

MISC. -0, -4, -C

	IMM 2	ZPAG 2	Z,X 2	(I,X) 2	(I),Y 2	ABS 3	A,X 3	A,Y 3
ORA	09	05	15	01	11	0D	1D	19
AND	29	25	35	21	31	2D	3D	39
EOR	49	45	55	41	51	4D	5D	59
ADC	69	65	75	61	71	6D	7D	79
STA		85	95	81	91	8D	9D	99
LDA	A9	A5	B5	A1	B1	AD	BD	B9
CMP	C9	C5	D5	C1	D1	0D	DD	D9
SBC	E9	E5	F5	E1	F1	ED	FD	F9

Op Codes Ends in -1, -5, -9, or –D

APPENDIX II AUTO STARTS

The most important reason to ROM your software is to have it up and running when the computer comes on. Commodore has made provision for this to happen on the C-64, but you must have the appropriate code in the appropriate locations on expansion ROM in order for this to happen.

You can auto-start machine code programs, BASIC programs, or programs that use both. BASIC programs will not run without the C-64 BASIC interpreter, so such programs, including an 80 byte auto start header, cannot exceed 8k in length. Machine code programs with auto start can be up to 16k in length. If a BASIC program is to be ROMmed along with some machine coded subroutines, the code subroutines should be written and tested at their desired finished location on ROM, that is somewhere in the $8000-9FFF block. Leave enough room below these routines to hold the 80 byte header plus the BASIC text, and do this development with the cartridge MODE switch in "0" position.

When BASIC programs are to be run directly from ROMs in $8000 to 9FFF they will have to be remapped to work there. This is because the first two bytes of every BASIC line (when crunched and stored in memory) are a pointer to the address in memory where the next BASIC line begins. This is covered in detail later.

We give two examples of auto-start headers, which must always be placed at address $8000. The first auto starts a machine coded program that begins at $8022. The second auto start header is designed to start a BASIC program which has been remapped to work at $8050.

Remember that all BASIC programs start with a single zero byte. Be sure you have a zero in the start address of the program.

Both Auto Start examples are designed to return smoothly to the C-64 BASIC operating system, upon encountering an RTS for which no return address was stacked in machine code, or a STOP or END instruction in BASIC.

These headers are given as examples, and they do not handle certain situations which may pop up. For instance, if your program requires allocation of RS-232 buffers, you may wish to set the top of memory pointer (use the KERNAL MEMTOP routine) or the top of BASIC during the auto start. Be sure to leave room for this type of code, if needed.

IIa Auto Start Header for Machine Coded Programs

```
$8000          bytes                  OA 80 9F FF C3 C2 CD 38 30 FF
$800A          LDX#$00                A2 00
               STX$D016               8E 16 DO
               JSR$FDA3               20 A3 FD
               JSR$FD50               20 50 FD
               JSR&FD15               20 15 FD
               JSR&E518               20 18 E5
               CLI                    58
$801C          JSR YOUR ROUTINE       20 22 80
               JMP (AOOO)             6C 00 AO
                                      $8021 contains last byte. your routine can
```

begin in $8022.

IIb Use this Procedure to put BASIC on EPROM for C-64

1) Develop and debug the BASIC text normally in the C-64. You may wish to save a copy on tape or disk.

2) Use Hexkit to enter the following Auto Start routine at $8000. Be sure the PROMQUEEN 64 is in 0 mode so RAM behind it is accessible.

```
$8000          BYTES                  OA 80 9F FF C3 C2 CD 38 30 FF
$800A          LDX#$00                A2 00
               STX$D016               8E 16 DO
               JSR$FDA3               20 A3 FD
               JSR$FD50               20 50 FD
               JSR$FD15               20 15 FD
               JSR$E518               20 18 E5
               CLI                    58
$801C          JSR$E453               20 53 E4
               JSR$E3BF               20 BF E3
               JSR$E422               20 22 E4
               LDA#$51                A9 51
               STA$2B                 85 2B
               LDA#$80                A9 80
               STA$2C                 85 2C
               JSRKEYBUF              20 33 80
               JMP$E39D               4C 9D E3
KEYBUFF $8033  LDY#$FF                AO FF
               INY                    C8
               LDA(8043),Y            B9 43 80
               BEQ05                  FO 05
               STA(0277),Y            99 77 02
               BNEF5                  DO F5
               STY$C6                 84 C6
               RTS                    60
```

TABLE $8043 RUN <RETURN> 00 (enter characters for "RUN RETURN" command to keyboard buffer with C/MODE) last byte is 8047

3) Transfer up the BASIC text from its starting point in the 64 at $800 to its new starting point at $8050.

a) Put the display at $8050
b) Type a T
c) Give the XFER FRM prompt the answer $800
d) Set the stop at high enough to copy all the BASIC text and hit <RETURN> again.

4) REMAP the BASIC next line pointers (one for each line) as follows:

a) Position display at $8051
b) Type an R for REMAP
c) Give the offset as 7850 and hit <RETURN>
d) Back up display two places to $8051
e) Find next line pointer to remap by typing a P
f) REMAP this with R again, use the same offset
g) Back up display 2 positions and type a P again h)
Continue remapping line pointers this way until you get to the 3 zeroes in a row that mark the end of the BASIC text.

5) Hit the RESET button to check that all this auto starts properly.

6) Hit RUN/STOP. Restart Autohex with SYS 29181.

7) Transfer the code from $8000 to whatever down to $2000 to whatever.

8) Burn EPROM with this text (FROM $2000 to whatever) . Use the EPROM in a socket that the C-64 address at $8000.

REMEMBER The C-64 does not permit you to have more than 8k of ROMmed BASIC text. Also, the first byte of a BASIC program must always be a zero. Be sure this is the case after your preparations.

lie Remapping BASIC Text with a BASIC Program

Once you have moved your BASIC text up to $8050 as per the example in Appendix IIb, you can use the following BASIC program (enter it normally) to remap the BASIC text of your original to work at the new location. This saves you the manual procedure using Autohex.

```
  5   REM:BASIC NEXT LINE POINTER ADJUSTMENT
  6   REM:ADDS OFFSET OF 30800 TO POINTER EACH LINE
  7   REM:REMAPS TEXT FROM $800 TO $8050 10   OF=30800 :
 OA=32849 15   NA=PEEK(OA)+256*PEEK(OA+1) : IF NA=0
       THEN 35
20   NA=NA+OF : H=INT(NA/256) : L=NA-H*256 25
POKE OA.L : POKE OA+1, H 30   OA=NA : GO TO 15 3
5   END.
```

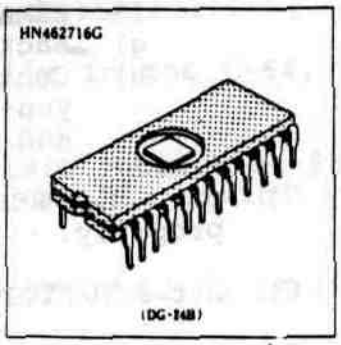
2716

HN462716G

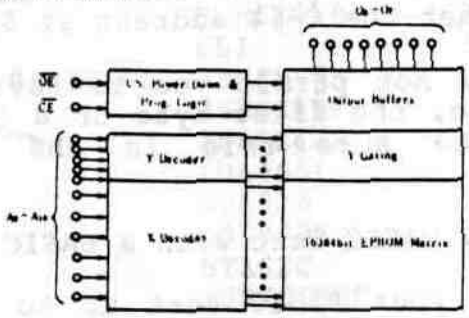
2048-word × 8-bit U.V. Erasable and Electrically Programmable Read Only Memory

The HN462716 is a 2048 word by 8 bit erasable and electrically programmable ROMs. This device is packaged in a 24-pin, dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern, whereby a new pattern can then be written into the device.

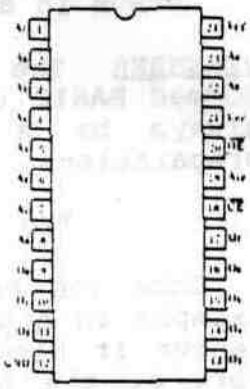
- Single Power Supply .... +5V ±5%
- Simple Programming .... Program Voltage: +25V DC
  Programs with One 50ms Pulse
- Static ..... No Clocks Required
- Inputs and Outputs TTL Compatible During Both Read and Program Modes
- Fully Decoded-on Chip Address Decode
- Access Time ..... 450ns Max.
- Low Power Dissipation .. 555mW Max. Active Power
  213mW Max. Standby Power
- Three State Output ..... OR-Tie Capability
- Interchangeable with Intel 2716



■ BLOCK DIAGRAM



■ PIN ARRANGEMENT



(Top View)

■ PROGRAMMING OPERATION

Pins / Mode	$\overline{CE}$ (18)	$\overline{OE}$ (20)	V_{PP} (21)	V_{CC} (24)	Outputs (9, 11, 13 – 17)
Read	V_{IL}	V_{IL}	+5	+5	Dout
Deselect	Don't Care	V_{IH}	+5	+5	High Z
Power Down	V_{IH}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IH}	+25	+5	Din
Program Verify	V_{IL}	V_{IL}	+25	+5	Dout
Program Inhibit	V_{IH}	V_{IH}	+25	+5	High Z

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Operating Temperature Range	T_{opr}	0 to +70	°C
Storage Temperature Range	T_{stg}	−65 to +125	°C
All Input and Output Voltages*	V_{in}	−0.3 to +7	V
V_{PP} Supply Voltage*	V_{PP}	−0.3 to +28	V

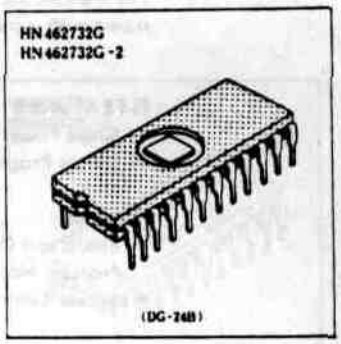
* With respect to Ground

2732

HN462732G, HN462732G-2

4096-word × 8-bit U.V. Erasable and Programmable Read Only Memory

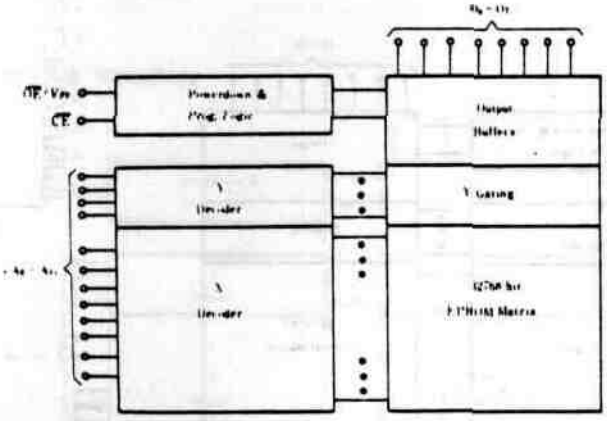
The HN462732 is a 4096 word by 8 bit erasable and electrically programmable ROM. This device is packaged in a 24-pin, dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern, whereby a new pattern can then be written into the device.



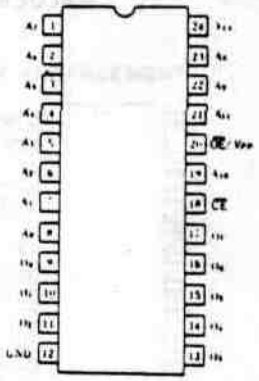
■ FEATURES

- Single Power Supply ..... +5V ±5%
- Simple Programming ..... Program Voltage: +25V D.C.
  Program with One 50ms Pulse
- Static ..... No Clocks Required
- Inputs and Outputs TTL Compatible During Both Read and Program Modes
- Fully Decoded On-Chip Address Decode
- Access Time ..... 450ns (max) HN462732G
  390ns (max) HN462732G-2
- Low Power Dissipation .... 150mA (max) Active Currents
  30mA (max) Standby Current
- Three State Output ..... OR-Tie-Capability
- Compatible with INTEL 2732

■ BLOCK DIAGRAM



■ PIN ARRANGEMENT



(Top View)

■ MODE SELECTION

Mode \ Pins	$\overline{CE}$ (18)	$\overline{OE}/V_{PP}$ (20)	V_{CC} (24)	Outputs (9~11, 13~17)
Read	V_{IL}	V_{IL}	+5	Dout
Stand by	V_{IH}	Don't Care	+5	High Z
Program	V_{IL}	V_{PP}	+5	Din
Program Verify	V_{IL}	V_{IL}	+5	Dout
Program Inhibit	V_{IH}	V_{PP}	+5	High Z

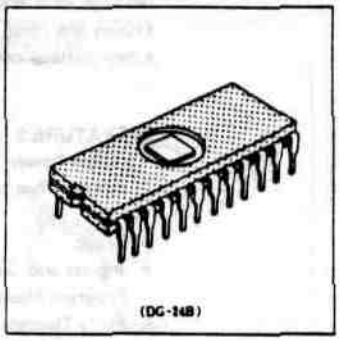
2732A

HN482732AG Series—Under Development –

4096-word × 8-bit U. V. Erasable and Programmable Read Only Memory

The HN482732A is a 4096-word by 8-bit erasable and electrically programmable ROM. This device is packaged in a 24 pin dual-in-line package with transparent lid.

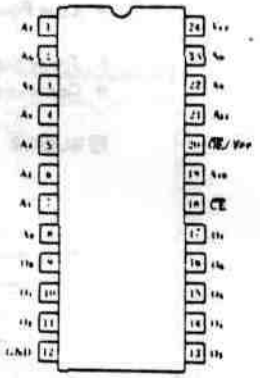
The transparent lid on the package allow the memory content to be erased with ultraviolet light.



■FEATURES

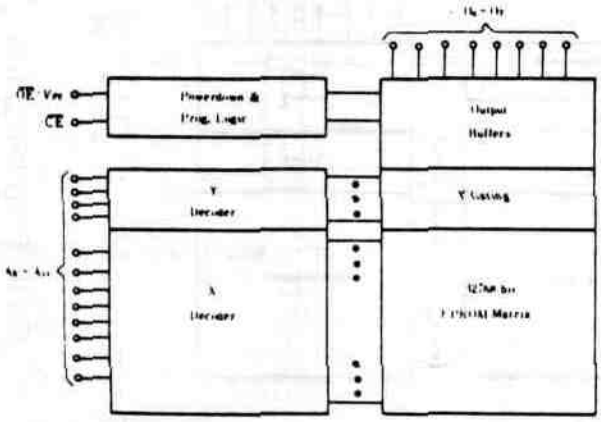
- Single Power Supply ..... +5V ±5%
- Simple Programming ..... Program Voltage: +21V D.C
  Program with one 50ms Pulse
- Static..... No clocks Required
- Inputs and Outputs TTL Compatible During Both Read and Program Mode
- Access Time ..... HN482732AG-20 200ns (max)
  HN482732AG-25 250ns (max)
  HN482732AG-30 300ns (max)
- Absolute Max. Rating of Vpp Pin ... 28V
- Low Stand-by Current ..... 35mA (max)
- Compatible with Intel 2732A

■PIN ARRANGEMENT



(Top View)

■BLOCK DIAGRAM

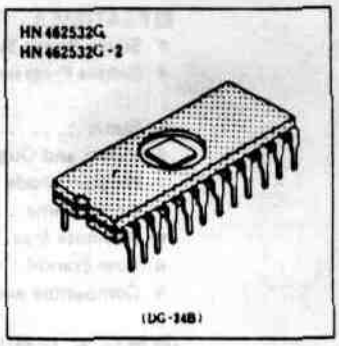


2532

HN462532G, HN462532G-2

4096-word × 8-bit U. V. Erasable and Programmable Read Only Memory

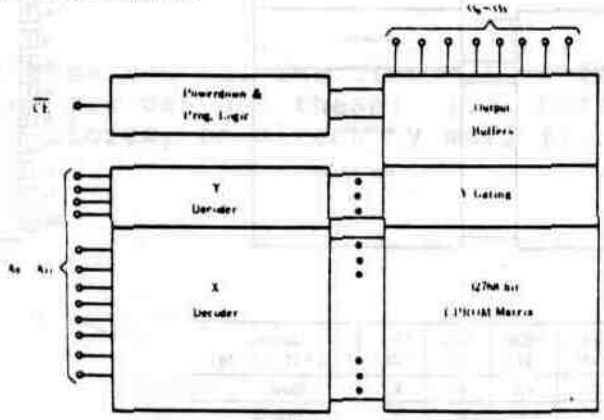
The HN462532 is a 4096 word by 8 bit erasable and electrically programmable ROM. This device is packaged in a 24-pin, dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern, whereby a new pattern can then be written into the device.



■FEATURES

- Single Power Supply ..... +5V ±5%
- Simple Programming ..... Program Voltage: +25V D.C.
  Program with One 50ms Pulse
- Static ..... No Clocks Required
- Inputs and Outputs TTL Compatible During Both Read and Program Modes
- Fully Decoded On-Chip Address Decode
- Access Time ..... 450ns (max.) HN462532G
  390ns (max.) HN462532G-2
- Low Power Dissipation .... 858mW (max) Active Power
  201mW (max) Standby Power
- Three Ststa Output ..... OR-Tie Capability
- Compatible with TMS2532

■BLOCK DIAGRAM



■PIN ARRANGEMENT

(Top View)

■MODE SELECTION

Mode \ Pins	$\overline{CE}$ (20)	V_{PP} (21)	V_{CC} (24)	Outputs (9 to 11, 13 to 17)
Read	V_{IL}	+5	+5	Dout
Stand by	V_{IH}	+5	+5	High Z
Program	Pulsed V_{IH} to V_{IL}	+25	+5	Din
Program Inhibit	V_{IH}	+25	+5	High Z

APPENDIX IV

Suggested Reading

1) C-64 Programmers Reference Guide Howard W. Sams Co., Inc.

2) MOS Microcomputers Software Manual Commodore Business Machines, Inc.

3) 6502 Machine Language Programming Lance Leventhal, Osbourne Press

4) 6502 Assembly Language Subroutines Lance Leventhal, Osbourne Press

5) Programming & Interfacing the 6502 Marvin DeJong, Blacksburg Press

6) 6502 Software Design Leo Scanlon, Blacksburg Press

7) Advanced 6502 Interfacing Leo Scanlon, Blacksburg Press

8) MOS Component Data Catalog MOS Division, Commodore Business Machines

New books come out all the time. Look for them at the same places you can get these: i.e. better computer & electronic stores, or direct by mail from the publisher,

WARRANTY & REPAIR

Each PROMQUEEN 64 CARTRIDGE is tested before it is shipped. The PROMQUEEN 64 is warrantied for a period of 90 days following purchase. If the PROMQUEEN 64 fails during this time, it will be repaired free of charge by Gloucester Computer Co., Inc., 1 Blackburn Center, Gloucester, MA 01930. You must send a copy of your sales slip and a written description of the fault. We'll send it back to you within two weeks. REMEMBER, THE PROMQUEEN 64 has many modes: check with your dealer, reread the manual or call us before you conclude that the cartridge is faulty.

The PROMQUEEN 64 CARTRIDGE is fully socketed and uses only top quality IC'S, resistors and capacitors. If it doesn't work, you don't have to solder to change the chips yourself. Chances are that a trace on the PC board could have broken if the cartridge were dropped too many times, or that something has caused a short. If you haven't used the cartridge for a long time, flick the switches (with the cartridge out of the VIC) a few times to clean their contacts.

KEEP THE CARD EDGE CONNECTOR CLEAN AND SHINY. If you see dirt on the connector, burnish it off with a clean pencil eraser.

GLOUCESTER COMPUTER CO., INC. ASSUMES NO LIABILITY FOR CONSEQUENT DAMAGES CAUSED BY MISOPERATION OF THE CARTRIDGE. BE SURE MATRIX SWITCH SETTINGS ARE CORRECT FOR THE EPROMS YOU ARE USING. AVOID RANDOM MATRIX SWITCH SETTINGS, AND CHANGE SETTINGS ONLY IN "0" MODE, OR WITH C-64 OFF.