UNIT-EK

UNIQUE TECHNOLOGIES

"EPRAM" EPROM Programmer

If you cannot - in the long run - tell everyone what you have been doing, your doing has been worthless.

— Erwin Schroedinger

A NOTE REGARDING ALL QED PROGRAMS BELOW VERSION 1.0 REVISION 2.0

The version of QED which you have received, although capable of programming 27128 EPROMS, does not contain the "EPROM" constant for this chip. This is because the standard programming method requires almost 15 minutes to program this very dense IC. We are currently developing a new version of QED which will utilize modified fast programming ala Intel, allowing the same chip to be programmed in a matter of minutes. This update package will be available to you for shipping/handling/media charges (translate "dirt cheap"); you will be notified when it is ready. If, in the meantime, you need to program a 27128, you may do so by inserting the "EPROM" constant in the OPT2 table. The current value is a "null" value of 0A40; you should substitute AA40. Data to assist you in this procedure can be found in the ADVANCED INFORMATION section of your EPRAM manual. You will also need to create a personality module; the module is identical to the 2764 module, with the addition of a strap from pin 4 to pin 15. Also, the .1 ufd capacitor is optional.

At the time revision 1 of the manual was last printed, TI was still expected to develop and release a 2528; they later announced that the 2528 was being scrapped. You may, therefore, ignore any information regarding 2528s in either the program or the manual. These references will be removed in the update package mentioned above.

Thank you for your continued support.

COPYRIGHT NOTICE

WARRANTY NOTICE

This manual and the associated hardware and software are sold AS IS, without warranty of any kind. We disclaim any warranties either expressed or implied, including warranties of merchantability and fitness for a particular purpose. Advertising claims made by us represent our honest opinion of the qualities and features offered by the products described; however, determining the suitability of the product for a given application is the sole responsibility of the purchaser. In no event shall Unique Technologies (UNITEK) be liable for consequential damages of any kind.

> Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which may vary from state to state.

All sales are made subject to the terms stated above. If these terms are not acceptable to the purchaser, then he should return this product in the original packing. Retention of this product by the customer shall constitute an agreement that he has read and accepts the terms of this statement.

Unique Technologies will make every attempt to correct any errors or defects brought to our attention, but this is only a statement of policy and not a warranty.

Manual Revision 1

TABLE OF CONTENTS

GREETINGS!

Thank you for your purchase of the Unique Technologies EPRAM system. We feel that this EPROM programmer is an excellent system at a fair price, and we hope you will agree.

This manual is your guide to the operation of the EPRAM system, and is divided into three main sections. The INSTALLATION GUIDE explains the basic operation of the system and (hopefully) provides enough information for you to easily configure EPRAM to your computer. Although you must be able to use an editor and assembler and be familiar with the I/O configuration of your computer, extensive knowledge is not required.

The USERS MANUAL provides the day-to-day information you will need to use the EPRAM system, and includes descriptions of all the commands. This is the section to turn to if you are having trouble remembering syntax or need to know how to perform a given operation.

The ADVANCED INFORMATION portion is for the true hardware and software hackers. In this section we have attempted to provide sufficient information for the experienced user to adapt the EPRAM system to his specific needs. Of course, the sofware source code and hardware schematics are also supplied for this purpose.

In addition, there are sections on troubleshooting, creating personality modules, and EPROMs in general. It is our hope that you will find this manual a useful reference tool.

If you have any suggestions for improvements to our EPRAM hardware or software, or find any bugs, we would greatly appreciate hearing from you.

Happy Programming!

A NOTE OF THANKS

We often hear complaints of microcomputer people raving about their systems while ignoring them when it comes to "useful work". We would like to go on record as having used our in-house SSB system and Alford and Associates' SCREDITOR III word processor to prepare this manual; I personally cannot imagine having done it any other way. SCREDITOR made things which would have otherwise been impossible a piece of cake. Thanks, John, for laboring so long over this program; it was worth every minute you put into it. And thanks to SSB for a fine system to run it on.

HARDWARE OVERVIEW

The hardware of the EPRAM system is contained on one 30 pin card, which may be installed at any available address in the host computer. A 28 pin ZIF socket accomodates both 24 and 28 pin EPROMs. An on-board voltage tripler develops the required programming voltages from the computer's power supply. Unlike many "hobby type" EPROM programmers, the EPRAM system is capable of continuous duty operation with no danger of overheating. LEDs monitor the programming and logic voltages applied to the EPROM socket. For maximum safety all pins of the EPROM socket are clamped to within one diode drop of ground (about .6 volts) when an EPROM must be inserted or removed. Personality modules are extremely simple and are built using inexpensive DIP headers.

SOFTWARE OVERVIEW

The hardware of the EPRAM system is supported by a menu-driven control program called QED. This software is extremely forgiving of errors and, short of overlaying it with data, is difficult to crash. The user is given the opportunity to abort most commands before operations begin. All commands are single characters, and the return key is seldom required. Input characters are automatically converted to upper case so the shift key can be disregarded. Feedback is always provided in the form of pertinent system information, "in progress" messages, current status indicators, and full English prompts. No data is ever allowed to scroll off the screen before the user is ready.

HARDWARE REQUIREMENTS

The EPRAM system is flexible enough to be adaptable to most computer environments. However, there are certain requirements which must be met if the system is to be installed without major modifications. Most of these requirements may be disregarded as desired, according to the skill and knowledge of the installer.

A disk-based 6800 or 6809 computer with a 30 pin I/O buss and the appropriate operating system.

An ACIA-based terminal, or any device or software which simulates an ACIA-based terminal.

A CRT or terminal capable of displaying 24 lines of 80 upper and lower case characters, minimum. Data should be accepted at a rate equivalent to 4800 baud or better. The display device must support home cursor, clear screen, and backspace functions.

At least 6K of user memory, plus room for the data to be programmed.

Note that the EPRAM system is not a load-and-go product, but will require adapting to ANY system on which it is installed.

BEFORE YOU BEGIN...

Make sure you have everything you ordered. For the full EPRAM package this includes the EPRAM board itself, a floppy disk, and the EPRAM manual. Take a moment to look over the hardware and check for any obvious shipping damage; also examine the disk to make sure it is not creased or crimped. You should pay special attention to this step if you see any evidence of damage on the outside of the package; a crushed box with half the packing hanging out is usually a good indicator that something may be wrong.

Also make sure you received the correct software... the right CPU, disk size, and operating system. After all, we do make misteaks now and then.

Finally, fire up your system and make a backup copy of the disk; then PUT THE ORIGINAL AWAY. Don't touch it again unless you destroy your copy. This is a good habit in general and can be a lifesaver; besides, lots of companies (us included) require that you send original disks back for updates and it's nice to know where they all are.

The EPRAM system consists of both hardware and software components. While installation of the hardware is a simple matter of selecting an I/O slot and plugging in the EPRAM card, the QED driver software must be configured for your system. To do this the source code supplied must be brought into your editor and the appropriate values installed. The source will then be assembled into the final object file.

This section provides the instructions needed for you to supply these values. In addition to these directions you will need certain information about your system; this data can usually be found in the manuals for your computer and peripherals.

One value, the DELAY constant, must be obtained before you can begin editing the source code. The next paragraphs are therefore appropriately titled...

DETERMINING THE "DELAY" CONSTANT

The EPRAM driver software uses a time delay constant called DELAY to develop a 50 mSEC programming pulse. This constant must be tailored to the clock speed of your system. This is done when the software is first assembled for your system, and needs never be modified, unless you change CPU clock speeds.

We have tried to make this process as simple as possible, and have developed four methods to determine the correct value of DELAY. The choice of which method to use depends on the computer in question and the equipment available.

Install the EPRAM board in your computer, then power up and boot your operating system. Using one of following methods, determine the DELAY value for your system. Once you have obtained this constant you can proceed to the next section.

METHOD #1: PRECALCULATION REQUIREMENTS: KNOWN CLOCK FREQUENCY

If your clock speed is known to be EXACTLY 1.0, 1.5, or 2.0 Mhz, you can use a precalculated value. Select the appropriate value from the table below and jot it down, then proceed to the next section. Skip methods two through four.

If you clock speed is EXACTLY:	Use this value for DELAY:
1.0 Mhz	$00FA
1.5 Mhz	$0177
2.0 Mhz	$01F4

NOTE: Just because your system is SUPPOSED to be running at a given clock speed does NOT mean that you are running exactly to frequency. If you have not checked your PHASE 2 frequency before, this is a good excuse to do so. If you are using an RC controlled clock, let your system run for a warmup period before taking any measurements.

METHOD #2: APPROXIMATION REQUIREMENTS: DIGITAL WATCH

This method has the advantage of not requiring any equipment except for a digital watch or clock. Assemble and run the program called SIGHT. You will be asked for the address of the I/O slot you have selected for EPRAM, then for a four digit timing value; you can select a starting value from the chart under method one. After entering this value the LV LED on the EPRAM board will begin flashing. The goal is to adjust the timing so that the LED blinks at a rate of exactly once per second. If the indicator is flashing faster than this, you need a larger value for DELAY; a slower rate means you need to decrease the value. Keep entering new values until the LED is blinking as close to a one Hz rate as possible. (The program looks at the terminal only after it finishes each blinking cycle, so there will be a short delay before the first digit entered is recognized. You should key the first digit, then wait for the program to echo it back before continuing. The last digit entered starts the new timing cycle, so you can synchronize the LED to your watch by keying the last digit exactly on the second.) You should ultimately compare the timing over a long interval, say 30 seconds or more. This method should get you within about 2% of the exact DELAY value; since most manufacturers specify an acceptable range of 10% for the program pulse this is quite adequate. When you are satisfied with the results, jot down the last number you entered, then exit the program and proceed to the next section. Skip methods three and four.

METHOD #3: CALCULATION REQUIREMENTS: FREQUENCY COUNTER

This method requires a frequency counter, and calculates the required DELAY value from known timing. Measure your exact PHASE 2 frequency on the buss, and use this formula to determine DELAY:

DELAY = FREQ / 4000

Jot the result down, then proceed to the next section. Skip method four. NOTE: The result of this calculation is in DECIMAL. All other methods result in a HEX value. Be sure to insert the value into the program accordingly!

METHOD #4: DIRECT MEASUREMENT REQUIREMENTS: OSCILLOSCOPE

This method requires an oscilloscope. Set your scope to 2V/DIV and 10 mSEC/DIV, then connect it to the PGM and GND test points on the EPRAM board. Now assemble and run the program called SCOPE. You will be asked for the address of the I/O slot in which you have installed the EPRAM board, then for a four digit timing value. This value can be selected from the chart shown under method one. When this value is entered a pulse train will appear on the scope. Note the time interval BETWEEN the narrow pulses. If this interval is less than 50 mSEC you need a larger value; likewise a smaller value is required if the time exceeds 50 mSEC. Key in new values until the positive portion of the waveform spans precisely 50 mSec. If you can't hit it exactly, get as close as possible. Jot down the value you last entered, then exit to your operating system. Now proceed to the next section.

MODIFYING THE QED SOURCE CODE

Now that the value for DELAY has been determined the QED source code can be adapted to your system. Pull out your system manuals and bring the source file QED.TXT into your editor. Immediately after the opening comments you will find the origin for the program; shortly thereafter is a section titled USER DEFINED VALUES. This section is the portion which must be modified; change the following values as required for your system. The values enclosed in parentheses may be left unchanged until a later date, if desired.

(PROGRAM ORIGIN) $0100

The program origin is controlled by the ORG statement at the start of the source code. The QED program may be placed anywhere in available memory as long as no conflicts are generated. For 6800 versions you should select a section of memory which will least often conflict with data to be programmed into your EPROMS. (Some users will wish to have two versions of the QED program, one at low memory and a second at high memory.) 6809 versions support a RELOCATE command which allows QED to be moved to any convenient location; in this case the ORG statement only controls the initial load location. Select an appropriate address and insert it in the ORG line.

EPRAM port #2 = $E008

This equate tells QED where you put the EPRAM board in your system memory map. Look up the FIRST ADDRESS of the I/O slot you have selected for the EPRAM board and key it into the EPRAM line.

ACIA port #1 = $E004

This equate is the first address of the ACIA which talks to your system terminal. QED talks directly to your terminal ACIA rather than using any OS I/O routines. Look up the FIRST ADDRESS of the I/O slot which contains your terminal ACIA board and insert it in the ACIA line.

NOTE: Video boards which simulate an ACIA, such as FEBE's VIDEO-PORT, can also be used with the QED program. PIA driven terminals and memory mapped video boards require user-supplied interface routines. (See page 4-11)

BACKSP $08

This value is the ASCII code recognized as a backspace by your terminal. Almost all terminals use 08 (control-H) but you may change this if required.

WARMON

This is your monitor's warmstart address. Unfortunately, many monitors do not document their warmstart addresses. If you have the source code for your monitor, you can locate the required entry point by finding the coldstart (RESET) address, then following the code past the point where the ACIA and stack are initialized. You should find a location just before the monitor prompt is issued which has a label. This should be it; try it and see. If all else fails use the coldstart address. (The warmstart address is preferred because the coldstart resets the ACIA, which usually interrupts the clear screen operation when exiting to the monitor. However, this is not a critical requirement.) Enter the appropriate address in the WARMON line.

WARMOS $CD03

This is your OS warmstart address, also known as the warmboot point. Insert the proper address in the WARMOS line.

CDFM $D403

This is the address of the file manager cleanup routine in your OS. In SSB's DOS™ this routine is called Close Disk File Manager (CDFM); in TSC's FLEX™ it is called File Management System Close (FMS CLOSE). Find the appropriate address and plug it into the CDFM line.

TCA $C100

This is the starting address of the transient command area supported by your OS. In SSB's DOS™ this buffer is called the Transient Command Area (TCA); in TSC's FLEX™ it is called the Utility Command Space (UCS). Find the appropriate address and place it in the TCA line. This value is not required for 6800 versions of QED.

DELAY 1 MHz = $00FA

This is the adjustable delay used by the timing subroutine. It must be adjusted to give a time inverval of 50 milliseconds when running with your system clock. Insert the value you obtained from the section DETERMINING THE "DELAY" CONSTANT (Remember that the CALCULATION method produces a DECIMAL value). If you do not know this value go back NOW and determine it.

LINES 24

This value is the number of lines available on your terminal, and is used to center information vertically on the screen. If your CRT displays more than 24 lines (the minimum), insert the number here.

(BUFST) $8000

This is the default starting location of the programming buffer. If you have a particular area of memory you will be programming often, you can set BUFST to the beginning of this section. BE SURE THE BUFFER DOES NOT OVERLAP THE QED PROGRAM!

(EPROM) 2716

This value determines which EPROM type the QED program defaults to; it comes set for the INTEL 2716. If you have a certain type of EPROM you will be programming often, you may want to preset the drivers to default to that type. Here is the procedure:

Look a little further in the opening section of the program, until you find a table marked OPT2. Notice that the table has an entry for each EPROM type. Each entry is composed of two lines. The first line is an FCC which tells you the EPROM letter and number (For example, the first EPROM is type A - a 2508). The second line is an FDB which indicates the control number. Scan down the table and find the default chip you want. Now refer to the number on the next line. Insert this number as the EPROM value.

HOMCLR $1B

This the sequence of ASCII characters which must be sent to your terminal to make the cursor move to the upper left corner and clear the screen. Almost all terminals, no matter how stupid, support the home and clear function. You can find this in the documentation for your terminal. Insert the required characters in the HOMCLR line seperated by commas, then add a null (00) at the end. You may use as many characters as required for your terminal, just be sure to remember the 00 at the end!

These are the only adjustments required unless you are using a system which does not meet the minimum requirements mentioned earlier. (If this is the case, you may have to modify QED considerably; it is up to you to know what you are doing!) Otherwise save the edited file and assemble it using the name of your choice (for example, PGM.$ for SSB's DOS™ or PROGRAM.CMD for TSC's FLEX™). If your file assembled properly, you are ready to begin learning your way around the EPRAM system. This is your cue to turn to the next section. See you there.

At this point you should have a functional EPROM programmer. (Congratulations!). In this section we will familiarize you with the basic operation of the system. If during the course of your initial explorations the results you get don't match what is listed here, you may wish to turn to the troubleshooting section.

ABOUT THE LV AND HV LEDS

The EPRAM board has two indicators marked LV and HV. LV stands for Low Voltage, and refers to the five volt logic supply to the EPROM socket. This supply is controlled by the driver program, and is turned off whenever the EPROM is not being accessed. At the same time the LV supply is turned off, QED clamps all of the pins on the socket to logic zero, which means they are within about one diode drop (or .6V) of ground.

The HV indicator monitors the High Voltage (programming voltage, or Vpp) for the socket. This voltage can be either 21 or 25 volts, depending on the type of EPROM being programmed. Again, this supply is controlled by the QED program.

Together the LV and HV LEDs monitor the status of the EPRAM hardware. With both indicators extinguished all lines to the socket have been clamped near ground and it is safe to insert or remove an EPROM, or change personality modules. (NEVER change EPROMS or modules unless BOTH of these indicators are extinguished! Although it is possible to get away with this, you will run the risk of damaging your EPROM.) When the program is accessing the EPROM for a read operation, the LV LED comes on. Both LEDs illuminate when a programming operation is in progress. The main thing to remember is never to fiddle with the EPROM or personality module unless both lights are OUT.

Now, if you will bring power up on your system you will notice that the LV LED illuminates. This is the normal noninitialized condition, and indicates that the card is receiving power. This state occurs any time the computer is turned on or a reset is applied.

Now run the QED program which you previously assembled (remember section 2?). Two things should happen immediately after the program finishes loading. First, you should see the LV LED extinguish; second, the screen should fill with the main QED menu.

THE QED BUFFER CONCEPT

One idea needs to be expanded upon before we begin discussing the menu. QED makes extensive use of a data area called the BUFFER; this is an area of memory which represents the EPROM being operated on. It is always the same size as the EPROM type selected, and exhibits a one-to-one relationship with it. (Thus the first memory location in the buffer corresponds to the first location of the EPROM, the second to the second, etc.) In fact, if the default programming options are selected, the EPROM will be programmed as an exact copy of the buffer area.

The buffer is maintained by the system for your convenience. Many commands require memory values before they can function, and in most cases the default limits representing the buffer can be used. Except for commands dealing directly with the EPROM you may select any limits convenient for your application; even limits outside the buffer area are acceptable. (Commands which read from or write to the EPROM calculate the chip addresses directly from the buffer addresses used. It is not possible to operate outside the buffer area with these commands.)

Playing with the buffer is one of the few ways you can get in trouble with QED. For example, it would be disasterous to place the buffer over QED, then load the contents of the EPROM into the buffer! Likewise, ignoring limits when writing to the buffer allows you to do all sorts of wonderful things, like overwriting your entire operating system with $3F's. QED will let you know when it senses impending doom, but DOES NOT STOP YOU from pulling the trigger. It is your responsibility to know what you are doing.

THE MAIN MENU

Now to the program. Your screen should be displaying a menu that looks something like this:

```
'EPRAM' EPROM Programmer Master Menu     Copyright 1982 - Unique Technologies

   680x Ver x.x, Rev x.x - EPROM type D: 2716 INTEL - Buffer $8000 to $87FF

                          K   Load buffer with constant
                          C    Examine or change buffer
                          M     Load buffer from memory
                          R       Relocate this program
                          D        Execute disk command
                          A         Alter buffer origin
                          H          Hex dump of buffer
                          S           Select EPROM type

                          V     Verify EPROM programmed
                          L      Load buffer from EPROM
                          E         Verify EPROM erased
                          P                Program EPROM

                          X              Exit to monitor
                          Q                   Exit to OS

Please select desired option:
```


The top line is composed of the menu title and the typical copyright blurb. The second line is considerably more interesting. It contains -- reading from left to right -- the program version and revision numbers, the EPROM type selected, and the current buffer limits.

This menu gives you complete control over the EPRAM system. A single keystroke is all it takes to get you into any of the commands provided, and by combining commands several other functions are available. The remainder of this portion of the manual is devoted to detailed descriptions of these commands and their use. You are encouraged to tinker with the commands as they are presented; about the only damage you can do is crash your system. But first...

RESPONDING TO ADDRESS PROMPTS

Most commands need an address or a pair of addresses to operate. Only valid hexadecimal digits are accepted; all other information is flagged by the terminal bell and rejected. All operations move from the beginning value to the ending value inclusive, and consider memory to be a closed loop (that is, $FFFF is followed by $0000). Thus using $8000 to $8002 (which will operate on three addresses) is NOT the same as using $8002 to $8000 (which would operate from $8002 through $FFFF, continue through $0000 and end at $8000!) If you select limits which are hazardous to the QED program itself, it will warn you by displaying the message "WARNING: Limits overlap program!".

Since the addresses desired will often be the buffer limits, QED provides a special W option which automatically inserts the appropriate value(s). W stands for Whole, and means you intend to use the whole buffer for your operation, or that you wish to start the operation at the beginning of the buffer. The K, C, M, H, V, L, E, and P commands all support the W option.

The requested addresses can usually be whatever values are convenient, and need not involve the buffer at all. However, the V, L, E, and P commands are exceptions to this rule. These commands deal with the EPROM and the addresses relate directly to it. Attempting one of these commands outside the prescribed buffer area would make no logical sense, and will generally only cause problems. These commands are seperated in the menu to remind you of this.

The K, M, R, V, L, E, and P functions give you a chance to skip out before operation begins. This is because most of these commands are potentially hazardous to memory or EPROM contents if incorrect information is passed to them. Also, there is no way to backspace or correct an entry once it is keyed in, so the abort function is essential. If you make an error when entering an address or data, you should key anything valid (all zeroes, for example) until the continue prompt appears, then abort and restart the command. (This is not as cumbersome as it sounds, since few keystrokes are usually involved.) The C, A, and H commands do not support the abort function since a minimum of information must be reentered if an error occurs.

Now, on to the commands!

K - LOAD BUFFER WITH CONSTANT

This command allows you to set any section of memory to a constant value.

When a K is entered QED prints a banner to indicate which mode you are in, followed by a line showing the current starting and ending locations of the buffer. (This is the standard format which most QED commands use.)

You are then asked for the first address which you want to set to a constant value. This location can be ANY ADDRESS in your entire system memory map, although it will usually be within the limits of the buffer area. (The question is worded to remind you of this.) Alternately you can specify the W option, which automatically sets the command to use the entire buffer area.

Next you must insert the last location you wish to set to a constant value. (If you used the W option in response to the beginning address question the ending value is preset and this question will be skipped.) Again, this value can be any location in your address space.

Finally, you are asked for the hex value you wish to insert in the block you have specified.

As an example, the following exchange will set the block of memory from $9000 to $9FFF to $3F's:

```
LOAD BUFFER WITH CONSTANT

Current buffer location $0000 to $07FF
Beginning buffer location (or W)? 9000
Ending buffer location? 9FFF
Value to be inserted? 3F

You may now C)ontinue or A)bort. C
```

Note that the area of operation was entirely outside the buffer limits; this is completely legal. For another example consider the following, which will clear the buffer area:

```
LOAD BUFFER WITH CONSTANT

Current buffer location $0000 to $07FF
Beginning buffer location (or W)? W
Value to be inserted? 00

You may now C)ontinue or A)bort. C
```

Note that the ending location prompt was not given, since the W option was selected.

C - EXAMINE OR CHANGE BUFFER

This command allows you to freely move through memory, examining and perhaps changing data in individual memory locations.

After the standard banner is printed, QED asks for the first address you wish to examine. If you want to begin at the first location of the buffer, you can specify the W option rather than an address.

Next the screen will clear and a prompt line will be displayed. Immediately below this line is displayed the selected address, followed by the contents.

At this point all the options listed in the prompt line are available. For example, to move to the next sequential memory address, hit the space bar. Likewise, a minus sign will move you back one memory location. Hitting the N key will cause QED to display the prompt "Address?", at which time you may enter a completely new address to examine. The Q key will return you to the main menu.

Finally, you may alter the data in the current location by entering its new value. You may do this either by entering any valid hexadecimal value, or by entering an apostrophe (') followed by an ASCII character. QED will convert either entry to binary and place it in the current memory location, then advance to the next address. If an attempt is made to write data to a ROM or a read or write only register, a question mark will be printed immediately after the data to indicate it was not read as written.

As data is entered the cursor will move down the screen until the bottom is reached. At this point data on the CRT will begin scrolling upward each time a new line is printed. Since this means the prompt line indicating your options will scroll off the top of the screen, a new one will automatically be displayed at the bottom.

The following example illustrates the use of the C command. All information in parenthesis are comments and not generated by QED.

```
                    EXAMINE OR CHANGE MEMORY

                Current buffer location $8000 to $87FF
                Beginning buffer location (or W)? 8315

        SPACE = forward  MINUS = back  N = new address  Q = quit  VALUE

8315 41                 (User types a space and advances location)
8316 00 4C              (User modifies data and advances location)
8317 7E -               (User types a minus to see modified data)
8316 4C Address? 8640   (User types N followed by a new address)
8640 41 'G              (User enters ASCII data and advances location)
8641 43 'O              (User enters ASCII data and advances location)
8642 0D                 (User enters Q to quit; returns to main menu)
```


M - LOAD BUFFER FROM MEMORY

This mode allows you to move blocks of data from one area in memory to another. Any size block of memory may be moved to either a higher or lower range of addresses, and the destination block may even overlap the source block. QED automatically determines how to copy the information so that no data will be lost.

After the standard banner is printed, QED asks you for the first buffer address you wish to work with. Since we would normally speak in terms of moving memory TO the buffer, this prompt refers to the start of the DESTINATION block, not the source. If you wish to move a block of memory which will fill the entire buffer, you can use the W option.

Next you should enter the last address of the destination block. This step is not necessary if you used the W option.

The final step is to key in the first address of the source block. QED will calculate the size and ending location of the source block automatically.

For example, suppose it is necessary to copy a large program into several smaller EPROMs. You can do this by copying EPROM-sized blocks of information to the buffer and programming one chip at a time. The exchange below will copy the first block of information, residing from $4000 to $47FF, to the buffer.

```
LOAD BUFFER FROM MEMORY

Current buffer location $0000 to $07FF
Beginning buffer location (or W)? W
Beginning source location? 4000

You may now C)ontinue or A)bort. C
```

Although it may seem confusing at first to specify the move in terms of the destination rather than the source, you will find that in most cases this works well, since you will be concerned with what information will fit in the EPROM rather than the limits of the source.

R - RELOCATE QED PROGRAM

This command allows you to move the QED program to any area of user memory. The move may be in either direction, and the new program may overlap the old copy if required. THIS COMMAND IS SUPPORTED ONLY IN THE 6809 VERSIONS OF QED.

QED responds to the R command by displaying the current locations of the buffer and program. It will then ask you for the new starting location for the program. Since QED would normally be relocated to move it out of the way of new data which is to be loaded in, no check for buffer overlap is made.

Due to the way QED handles the relocation, you do not have to be concerned about whether the new location overlaps the old program, or in which direction you are relocating. However, you should be conscious of your destination and any data you might destroy. Also, attempting to relocate QED to non-existent memory or ROM would be disasterous.

The following example shows the QED program being moved from its default load location to a block starting at $4000, perhaps to make room for data which must load at $0000.

```
RELOCATE QED PROGRAM

Current buffer location $8000 to $87FF

Program is located from $0000 to $162C
New program start? $4000

You may now C)ontinue or A)bort. C
```


D - EXECUTE DISK COMMAND

The D command provides an interface to the computer's operating system, and allows you to execute most OS commands.

In response to the D command, QED will clear the screen and print a banner indicating that it still has control. Immediately below will be displayed your operating system prompt; at this time you may enter any operating system command exactly as you normally would. The command will execute, then the prompt "Hit any key to continue" will be displayed. QED returns to the main menu when any character is entered in response.

Caution should be exercised when using this command. While most OS commands execute in a dedicated "transient command area", some are too large to fit, and reside in other portions of memory. If a command were to load or execute over the buffer you could lose valuable data; if the QED program itself were overlayed your system would crash as soon as the OS command finished execution.

The D command has a wide range of uses, such as examining disk directories, loading and saving binary files for programming, debugging program segments, etc. The example below shows a directory listing on an SSB system.

Executing DOS command under control of QED

```
DOS: DIR,1/L

 FILENAME PROT FT FS BTBS ETES SIZE   FILENAME PROT FT FS BTBS ETES SIZE

COPY  .$   L   BS OK 814A 8153   10  EPRAM9.$   L   BS OK A64D C44C   23
EPRAM9.TXT  DW CS OK 8655 A64C  177  SCOPE9.$       BS OK A04F A050    2
SCOPE9.TXT  DW CS OK A043 A04E   12  SIGHT9.$       BS OK A747 A748    2
SIGHT9.TXT  DW CS OK A655 A746   12  TRANSF.TXT     CS OK A841 A842    2

8 files total, using 2 to 177 sectors each. 250 sectors used, 1748 free.

Hit any key to continue.
```

A - ALTER BUFFER LOCATION

The A command allows you to move the buffer to any location in your system memory map.

QED responds to this command by displaying the current buffer limits and asking where you wish the new buffer to start.

By using this command it is possible to move the buffer to the data, rather than moving the data to the buffer. It also allows several otherwise impossible operations, such as loading the contents of the EPROM to any block of memory.

It is entirely possible to move the buffer to a position which partially overlaps or totally engulfs the QED program itself. Since QED makes no assumptions about what you are trying to do, no warning is given. However, any mass-change operation (K, M, L) attempted on the buffer after this point will elicit the message "WARNING: Limits overlap program!".

The following example shows the buffer being moved from $8000 to $A000:

```
ALTER BUFFER LOCATION

Current buffer location $8000 to $87FF
New buffer start? $A000
```

H - HEX DUMP OF BUFFER

This command generates a HEX/ASCII page dump of any section of memory.

QED will respond to the H command by displaying the current buffer limits, then asking where you wish to start the dump. If you wish to begin at the first page of the buffer, you may use the W option.

At this point QED will clear the screen and begin the dump by displaying the memory page which contains the address specified. THE DUMP ALWAYS BEGINS AND ENDS ON PAGE BOUNDIES. Thus if you specified location $8143, QED would dump $8100 to $81FF. This method was found to be superior to the standard practice of starting and ending wherever specified; rather than having to count bytes to find a given address, the QED page dump command ALWAYS displays each byte of a page in exactly the same location, with full address pointers.

The dump is displayed in a fairly standard format, with an ASCII representation of the data to the right of the HEX display. Values without printable ASCII equivalents are represented by a period.

Below the dump is an options list and prompt. For your convenience, the option mnemonics are identical to those used in the C command. The space bar moves the display forward to the next memory page, and the minus key moves the display back to the previous page. Entering an N causes QED to prompt for a new dump address. Alternately, you may enter the address directly without using the N key (this is the meaning of the VALUE prompt).

As an example, the following exchange shows a user dumping two pages. To conserve space only the first and last lines of each dump are shown.

```
                    HEX DUMP OF BUFFER

                Current buffer location $8000 to $87FF
                Beginning location (or W)? W

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
     -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
8000  13 15 14 20 12 11 10 07 08 07 04 04 04 00 00 00  ... ............
80F0  7E EA B3 7E EA 5C 7E E8 06 7E E8 06 0B 11 50 03  ~..~.\~..~....P.

SPACE = forward  MINUS = back  N = new address  Q = quit  VALUE  4327


      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
     -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
4300  5F 1C FE 39 7F DE 49 E6 03 C1 03 22 D3 8E DE 30  _..9..I....".....0
43F0  84 30 44 44 44 44 8E DE 38 C6 04 8D 11 35 02 84  .0DDDD..8....5..

SPACE = forward  MINUS = back  N = new address  Q = quit  VALUE  Q
```

S - SELECT EPROM TYPE

This command is used to configure QED for the desired EPROM type.

In response to this command, QED will clear the screen and display the second of its two menus, which looks something like this:

EPROM Selection Menu Copyright 1982, Unique Technologies

680x Ver x.x, Rev x.x - EPROM type D: 2716 INTEL - Buffer $8000 to $87FF

8K EPROMS (1K BYTE)		16K EPROMS (2K BYTE)	
A	2508 TI	C	2516 TI
B	2758 TI/INTEL	D	2716 INTEL

32K EPROMS (4K BYTE)		64K EPROMS (8K BYTE)	
E	2532 TI	H	2564 TI
F	2732 INTEL	I	2764 INTEL
G	2732A INTEL		

128K EPROMS (16K BYTE)	
J	2528 TI
K	27128 INTEL

SELECT DESIRED EPROM:

This menu displays a complete list of all the EPROM types programmable by the EPRAM system, and assigns a letter to each one. To select the desired EPROM, simply locate it on the menu and enter the associated letter. The main QED menu will return with the new EPROM type listed in its header, and the buffer limits adjusted accordingly.

Note that each EPROM is followed by either TI or INTEL. Due to a rather involved situation early in the EPROM saga, quite a bit of confusion was generated by the numbers manufacturers assigned various EPROMs. For example, INTEL and TI produced identical triple voltage 2708s and single voltage 2758s. However, INTEL's 2716 is a single voltage EPROM, while TI's 2716 is a totally incompatible triple voltage version! (TI's number for a single voltage 2K byte EPROM is 2516.) Thus it became imperative for a time to mention not only the chip number but the manufacturer, to be sure you and the distributor were both talking about the same chip!

Manufacturer's literature will usually equate their EPROMs to one of these two (I.E., "Compatible with INTEL 2716") when referring to pinout and programming requirements. By using this data it is possible to program almost anyone's EPROM, simply by selecting the appropriate equivalent in the EPROM selection menu.

Also, it should be mentioned that several of the chips listed are identical as far as QED is concerned. For example, the 2732 and 2732A are both programmed in exactly the same way. However, for the sake of completeness and esthetics we have assigned each EPROM its own letter in the menu.

Finally, remember that SELECTING AN EPROM BY THE MENU IS ONLY HALF THE JOB. The menu selection sets up the QED software properly, but you must also insert the appropriate header in the personality module socket to configure the EPRAM hardware.

V - VERIFY EPROM PROGRAMMED

The V command causes QED to compare the contents of the buffer with the contents of the EPROM and report any mismatches.

In response to the V entry QED will display the current buffer limits, then ask you for the first location you wish to verify. THE EPROM ADDRESSES ARE DIRECTLY RELATED TO THE BUFFER ADDRESSES GIVEN. VERIFICATION SHOULD NOT BE ATTEMPTED OUTSIDE THE BUFFER AREA. If you wish to verify the entire EPROM against the buffer, you may use the W option.

Next QED will ask for the last location to verify. Again, this address should be within the buffer limits. This prompt will not be displayed if you used the W option previously.

QED then examines each buffer location within the limits specified, comparing the contents with the corresponding locations in the EPROM. During this process the banner "Verification in progress" will be displayed (although it may remain on the screen for a very short time). If the contents of the EPROM agree with the data in the buffer you will be returned to the main menu without comment. If an error is encountered the terminal bell will be sounded and the message "Verify error at buffer address $XXXX, expected $YY, read $ZZ" will be displayed. XXXX is the buffer address where the mismatch was found, YY is the data in the buffer QED was expecting, and ZZ is the data actually read from the EPROM. At the end of a verification with errors the data will remain on the screen and the message "Verification completed. Errors detected. Hit any key to continue" will be displayed. Touching any key will return you to the main menu. If more errors are detected than will fit on the display, the message "Verification failed. Operation terminated. Hit any key to continue" will be displayed; again, touch any character to restore the main menu.

The following example shows the second half of a 2K EPROM being verified against the buffer, with no errors resulting:

```
VERIFY EPROM PROGRAMMED

Current buffer location $8000 to $87FF
Beginning buffer location (or W)? 8400
Ending buffer location? 87FF

You may now C)ontinue or A)bort. C
```

L - LOAD BUFFER FROM EPROM

This command allows you to copy the contents of the EPROM to the buffer area.

After the standard banner is displayed, QED asks for the first buffer address to operate on. THE LIMITS SPECIFIED FOR THIS COMMAND RELATE DIRECTLY TO THE EPROM BEING OPERATED ON, AND SHOULD BE CONFINED TO THE BUFFER AREA. It is not possible to load the EPROM into memory outside the buffer area. To do this you must move the buffer (see the A command). To copy the entire EPROM to the buffer you can use the W option.

Next you should enter the last address to operate on. REMEMBER THAT THIS VALUE SHOULD BE WITHIN THE CONFINES OF THE BUFFER. If you selected the W option this question will not be asked.

As an example, the following dialogue will move the SECOND HALF of a 2K byte EPROM into the corresponding section of the buffer:

```
LOAD BUFFER FROM EPROM

Current buffer location $0000 to $07FF
Beginning buffer location (or W)? $0400
Ending buffer location? $7FFF

You may now C)ontinue or A)bort. C
```

Note that there is no specification as to what portion of the EPROM is to be moved to memory; this information is implied in the buffer values given. Remember that all commands dealing directly with the EPROM exhibit a one-to-one relationship between the buffer and the EPROM addresses.

E - VERIFY EPROM ERASED

This command checks the section of EPROM specified to make sure it is properly erased.

QED responds by displaying the current buffer limits, then asking for the first location you wish to verify. THE EPROM ADDRESSES ARE CALCULATED FROM THE BUFFER ADDRESSES GIVEN; YOU SHOULD NOT ATTEMPT A VERIFICATION OUTSIDE THE BUFFER AREA. If you which to verify that the entire EPROM is erased, you may use the W option.

Next QED asks for the last address you wish to verify. Again, this value should be within the buffer area. If you used the W option previously, this prompt will not be displayed.

QED examines all EPROM locations within the limits specified, checking to see that each address contains an $FF. ($FF is the normal "unprogrammed" state of all UV eraseable EPROMS). During this time the banner "Verification in progress" will be displayed (although possibly for a very brief period!). If no programmed locations are found, you will be returned to the main menu without comment. If an error is located, the terminal bell will sound, and the message "Verify error at buffer address $XXXX, read $ZZ" will be displayed. XXXX is the buffer address corresponding to the EPROM location where the error was found, and ZZ is the data read from the EPROM. At the end of a verification with errors the data will remain on the screen and the message "Verification completed. Errors detected. Hit any key to continue" will be displayed. Touching any character on the keyboard will return you to the main menu. If more errors are detected than will fit on the display, the verification process will be aborted, and the message "Verification failed. Operation terminated. Hit any key to continue" will be displayed; again, touch any character to restore the main menu.

As an example, the following exchange will verify an entire EPROM to be sure it is properly erased:

```
VERIFY EPROM ERASED

Current buffer location $8000 to $87FF
Beginning buffer location (or W)? W

You may now C)ontinue or A)bort. C
```

P - PROGRAM EPROM

The P command allows you to program an EPROM with any portion of the buffer contents (which after all is what this is all about, right?).

QED responds to the P command by displaying the current buffer limits, then asking for the first location you wish to program. THE EPROM ADDRESSES ARE DETERMINED FROM THE BUFFER ADDRESSES AND SHOULD BE CONFINED TO THE BUFFER AREA. If you wish to program the entire EPROM you can use the W option.

Next you should supply the last address you wish to program. Again, this value should be within the limits of the buffer. If you used the W option as your response to the beginning address question this prompt will not be displayed.

The programming process is a time-consuming procedure. Each location of an EPROM requires a 50 millisecond pulse to program, so with a 2K byte chip this is a minimum of about 1.7 minutes, not including program overhead. To assure you that your computer has not crawled off into a corner and quietly died, QED displays a banner showing the current address during the programming process. The addresses are incremented by sixteen with the least significant nybble zero, and refer to the buffer values.

After the programming phase is completed, QED automatically performs a V command - verify EPROM programmed - on the programmed section. If no errors are detected you will be returned to the main menu without comment; otherwise the errors will be displayed as explained in the V command description.

The following example programs an EPROM with the contents of the buffer:

```
PROGRAM EPROM

Current buffer location $8000 to $87FF
Beginning buffer location (or W)? W

You may now C)ontinue or A)bort. C
```


X - EXIT TO MONITOR / Q - EXIT TO OS

These two commands are used to exit the QED system. The X command hands control to the system monitor (usually a low-level debugger and I/O processor such as MIKBUG), while the Q command returns to the operating system.

The X command may be used to "get out of" QED to perform some rudimentary function prior to programming an EPROM. When the X key is hit, QED will home the cursor and clear the screen. Your monitor prompt should then appear in the upper left corner of the display.

The Q command is normally used to exit QED when a programming session has been completed. Alternately, in cases where you have a large number of OS functions to execute and typing the D command before each would be awkward, you could instead skip out of QED, perform your operations, then re-enter. Some OS's save the last transient vector used, and with these you can restart QED by performing a simple "GO" command or equivalent; otherwise, a "jump to address" command will be required. (This of course assumes QED was not destroyed by any of your OS commands, or relocated from its original load position.)

SOME EXAMPLES

Now let's run through a couple of sample sessions to help give you a feel for the operation of the system. In both examples we will define a problem and present one of the ways it can be dealt with. We will assume that EPRAM is installed on an SSB DOS based system, QED was assembled with an ORG of $0000, the default buffer starts at $8000, and the default EPROM is a 2716.

PROGRAMMING FROM SOURCE CODE

Let's assume you have a program named UCT.BIN that you are sick and tired of loading into RAM and want to blast into EPROM once and for all. You have a slot open for a 2716 at $E800 to $EFFF, and the program will fit nicely in 2K of space. You have the source code.

The first steps are really outside the realm of this manual, but should be pointed out at least once. First of all, make sure the program is ROMable. This means no self modifying code, no variable storage in the middle of the program, etc. The safest method for variables is to store them on the stack, although you can use scratch RAM when available.

Next assemble a version of the program ORG'ed for the destination (in this case, $E800); the resulting object file will be stored on disk.

Now you have a choice of running QED or moving the binary file into memory first. If you run QED, use the D command before the next step.

LOAD0 FILE.BIN,8000

At this point you have a nice tidy binary file ready to load at, unfortunately, $E800. Since it must be moved to RAM it will have to be loaded with an offset. The QED buffer can be placed anywhere, so let's pick an simple offset. One fact which is easy to remember is that adding $8000 inverts the MSB of an address; thus $E800 would become $6800, a very useable (and for some reason particularly charming) address. So, let's load the binary file with an $8000 offset. Your OS file loader would be used here.

Now you can move the buffer from its default load location to the data using the A command. (Alternately, you could have calculated the offset so the data loaded exactly into the buffer area, or you can use the M command to move the file to the buffer once it is loaded.) Since the program was assembled to run in EPROM, no further manipulation is required.

The EPROM being used happens to be the default type, so you do not need to use the SELECT EPROM option. However, if you work with more than one type of chip, it would pay to make sure you have the right module installed on the EPRAM board.

Now you should insert the EPROM in the programming socket, being sure to remember the orientation rules (chip all the way to the left, pin one in the upper right corner). Then you might want to verify that the EPROM is erased before continuing; you would use the E command here. After this there is nothing left but to program the EPROM (command P).

COPYING AN EPROM WITH CHANGES

In this example, you have a 2732 EPROM containing a monitor program you would like to use. Unfortunately, it uses the wrong I/O address. You have disassembled enough of the object to know that all the references are made through one constant, and that simply changing that value will do the trick.

The first step is to load QED, so the EPROM socket will be made safe for the insertion of the EPROM.

Now you can insert the EPROM to be copied into the EPRAM board socket. (Alternately you could place it in an empty socket in your system somewhere and move the buffer to that location. This would eliminate the read-in step.)

At this point it is necessary to tell your programmer that you are not using the default 2716 chip. Configure the EPRAM board by removing the 1-2K personality module and inserting a 2732 unit. (See the section on personality modules for more information). Don't forget to use the S command to tell QED about the new chip also.

The next step would be to move the contents of the EPROM to be copied into the buffer, using the L command. After this step you can make whatever changes are required to the data in the buffer with the C command. (You can also use the "D" command with your OS to save a copy of the code in a disk file for later use).

Now the buffer is ready to be programmed into a new EPROM. Remove the original EPROM from the EPRAM socket and substitute a new one. As in the first example, it is a good idea to make sure the new chip is completely erased, using the E command. And now you are ready to program once again, using command P.

Multiple copies could be made, simply by continuing to insert blank EPROMs and repeating the verify erased and program commands.

INSERTING AN EPROM

Now that we've covered the commands, let's discuss the hardware again for a moment. Near the top of the EPRAM card is the ZIF EPROM socket. ZIF stands for Zero Insertion Force, which means that you never have to worry about bent pins and such when plugging in your EPROMs. Notice the little lever on the upper right hand corner of the socket; it has two functions. One of them deals with the socket itself. When this lever is in the "UP" or "OUT" position, the socket is open. To insert an EPROM, place it in the socket with the lever in the UP position. Now, while holding the EPROM in place flip the lever to the "DOWN" or "IN" position. This locks the EPROM in place, and you probably couldn't get it out now if you wanted to. To remove the chip, place a thumb on the EPROM or place your hand under it (because when you release the lever it will literally fall out) and flip the lever back to the UP position.

ORIENTING THE EPROM

Yes, we did say the lever has two purposes. The other one deals with the orientation of the EPROM. You may have noticed that the socket has 28 pins, whereas the majority of EPROMs have 24. This means that, out of six possible ways to insert the smaller chips, five are wrong (which is not as many ways as there are to insert a disk incorrectly)! This is where the lever comes in; it indicates the corner of the socket in which you should place pin one of your EPROM. Also notice that a small arrow is printed on the printed circuit board under the opposite end of the socket. This arrow is a reminder that 24 pin EPROMs should always be seated as far to that end of the socket as possible.

In other words, always place an EPROM in the socket as far to the left as possible, with pin one in the upper right hand corner, near the lever. See the diagram in the appendix for more information.

INSTALLING PERSONALITY MODULES

Immediately below the EPROM socket is a sixteen pin socket marked "PERSONALITY MODULE". This is where, by coincidence, you insert the personality module for the EPROM you are programming. Pin one of the module is placed in the upper right hand corner of the socket, exactly as with the EPROM (in fact, ALL ICs on the EPRAM board plug in with the same orientation.) Pin one is also marked on the circuit board as a small dot.

To save expense, and because the module will not be changed as often as the EPROM, an ordinary socket is used here. You should therefore use a certain amount of caution when inserting a module, so as not to damage the pins.

You're probably here because you want some specific technical information. Well, this is the place to be. Be warned, though, that this section simply presents the facts; you will have to decipher them for yourself!

THE BIRTH OF EPRAM

The EPRAM programmer was originally designed for in-house use on several of our own projects. We wanted a programmer which was versatile enough to program any of the single voltage chips we might have to use. Triple voltage EPROMs we dismissed, since they will not be included in any new designs. (In fact, even the relatively old SWTPC MP-A2 used single voltage 2516s!) The thought foremost during the development was to make the hardware as simple and inexpensive as possible, using readily available parts. When the project was completed we realized that out little proto-kluged programmer could do most of what the $200 plus machines could do, at a fraction of the cost. The typical "better mousetrap" fever caught us, and we moved EPRAM into production.

A WORD ABOUT OUR EPROM PIN NUMBERING...

Fortunately for EPROM users, the manufacturers were able to agree on a certain amount of standardization for pin useage. In fact, when chip density became too great for a 24 pin package, manufacturers kept the old EPROM pinout, simply adding four pins at the top of the package. Thus pin one on a 24 pin EPROM serves the same purpose as pin three on a 28 pin package, pin 24 is the same as pin 26, etc. See the diagram in the appendix for more information.

To avoid the double pin numbering made necessary by this change, we have developed a simple system for referring to EPROM pin numbers, and use it in both this manual and the hardware schematic. All the pins which would exist on a 24 pin EPROM are referred to by their 24 pin equivalents, regardless of actual IC size. The four pins which do not exist on a 24 pin chip are referred to by their real numbers, followed by an asterisk to indicate that they are 28 pin numbers. The pin numbers for a 28 pin EPROM thus are 1*, 2*, 1, 2, 3... 22, 23, 24, 27*, 28*. Therefore, pin nine is forever the D0 line, regardless of the chip size.

EPRAM CONTROL AND DATA LOGIC

The EPRAM logic consists of a 2 MHz 6821 PIA, a pair of 74LS273 Octal D Latches, and associated discrete hardware.

The 6821 provides the interface to the system's S30 I/O buss. Both ports of the PIA are fully utilized, with only the interrupt inputs CA1 and CA2 free.

The B port of the PIA connects directly to the EPROM socket, and is permanently assigned to the data lines (pins 9 thru 11 and 13 thru 17). The B port is switched between input and output modes in software.

The A port, on the other hand, is always configured as an output. It is routed to the paralleled octal latches, which selectively capture data from the port. This method effectively doubles the number of A port lines available.

The CA2 output of the PIA serves two functions. It is used as a clock to latch data into the "lower" of the two latches (lines A0 thru A7), and also controls the low voltage to the EPROM socket. This line is thus labeled LVEN, for Low Voltage ENable.

Similarly, the CB2 output controls the "upper" latch (A8 thru A13, CS and PGM) and the high voltage to the socket; this line is therefore called HVEN for High Voltage ENable.

Data is read from the EPROM simply by reading the B port. Similarly, data is transmitted by switching the B port to output mode, then writing data to the port.

Addresses and control signals, however, must be transmitted by a multi-step process. First, the data for the lower latch is sent to the A port, then the LVEN is pulsed to latch the data. Next the data for the upper latch is sent to the A port, followed by a pulsing of the HVEN line. (The actual latching may also be performed in reverse order; i.e., upper then lower.)

The remainder of the logic is devoted to switching the EPROM power supplies. Referring to the schematic diagram, the LVEN line drives TR2, which in turn controls TR1, the actual switching transistor for the 5V EPROM supply. In a like fashion the HVEN line controls TR4, which drives TR3, the high voltage switching transistor. HVEN also switches the 555 on and off, so that the high voltage circuit does not run unless the supply is actually required.

EPRAM VOLTAGE SOURCES

The EPRAM board contains four regulators. These regulators, in conjunction with the 555 and associated circuitry, provide all the necessary voltages for the card and EPROM.

IC4, the "system" regulator, is a 7805 which provides the 5 volt supply for the 6821 and the 74LS273 latches; it is on at all times.

IC2 is the 5 volt regulator for the EPROM socket. The unregulated 8 volts from the computer's I/O buss passes through the low voltage switching circuitry of TR1 and TR2 and is regulated to 5 volts by IC2. Its output is connected to the EPROM socket's pin 28*, as well as pin 13 of the personality module socket. Power is only applied to IC2 during access to the EPROM socket or after initial power-up or a reset.

(The alternative to using a seperate regulator for this function would be to switch the 5V regulated voltage from IC4. However, there is roughly a .6V drop across TR1, which would result in a Vcc of 4.4 volts for the EPROM. By

placing the switching network before a dedicated regulator a full 5 volt Vcc is assured.)

IC1 supplies the Vcc for the 555 timer. The unregulated 18 volt supply from the system buss is clamped to approximately 12.6 volts due to the forward voltage drop across D1 in the 7812's ground leg. (Incidentally, this diode is a 1N4001 or better; all other diodes on the EPRAM board are 1N914s or equivalent.)

This 12.6 volt source feeds the 555, which is configured as an astable multivibrator producing a near-square wave at roughly 15 KHz. This square wave has an amplitude of approximately 11 volts, due to limitations within the chip.

This square wave is fed to a voltage tripler comprised of capacitors C9 through C12 and diodes D3 through D6. When the output of the 555 goes low C9 is charged to approximately 12 volts through D3. When the 555 output returns high D3 is reverse biased, and the charge on C9 (which is now in series with the output from the 555) charges C11 through D4. The resulting voltage on C11 is about 22 volts. The second half of the voltage tripler operates in a similar fashion, but instead of charging C10 from 12.6 volts, the increased voltage from C11 is used. With a Vcc of 12.6 volts from IC1 the unclamped output across C12 would be approximately 30 to 35 volts.

(Due to the combination of pre-regulator IC1 and the voltage tripler EPRAM can operate with "+18" supplies ranging from about 15 volts to well over 30 volts with no change in performance. Many programmers which use a voltage doubler have a more restricted operating range.)

D7 is a five watt zener which clamps the voltage across C12 to 30 volts maximum. This zener protects C12 and IC3 from the excessive voltage which could be developed if the high voltage were enabled with no EPROM in the socket.

The output of the tripler then passes through the high voltage switch TR3 to IC3, an LM317 variable voltage regulator. The output of this IC is either 21 or 25 volts, depending on the divider connected to the IC's reference pin. R13 provides a 21 volt Vpp; R14 adds an additional four volts for 25 volts total. R13 and R14 are paralleled by trim resistors TL (Trim Low) and TH (Trim High) respectively; these resistors are factory selected for precise Vpp values. The normal Vpp is 25 volts; bypassing R14 with a personality module strap produces the 21 volt supply.

The low voltage supply is controlled by the LVEN line, which drives the switch pair TR1 - TR2. Likewise, the HVEN line controls the TR3 - TR4 switch. In addition, the HVEN line is also tied to pin 4 of the 555, which acts as an enable line for the oscillator. Thus the voltage tripler operates only when the high voltage is needed. Capacitors C13 and C14 in the high voltage circuit and C4 in the low voltage circuit serve to hold the voltages up briefly after the associated supplies have been turned off. This allows the program to use the HVEN and LVEN lines to strobe data into the octal latches without affecting the voltages supplied to the EPROM socket.

While the LV LED is connected directly to the five volt EPROM supply, the HV LED is driven by the collector of the high voltage switch driver TR4; this is to avoid unnecessary loading of the voltage tripler. Diode D8 prevents the low reverse breakdown voltage of the HV LED from affecting the switching characteristics of the TR3 - TR4 pair.

The EPRAM hardware is overdesigned to a certain extent. This is evidenced in part by the voltage tripler where a voltage doubler probably would have been acceptable. The use of TO-220 regulators is another example; the TO-92 versions are good to 100 mA and would have been quite satisfactory. However, by using the heavier regulators we have eliminated all concern about duty cycle limitations and overheating; the EPRAM system is capable of operating continuously 24 hours a day. In fact, although the largest current demand is about 30 mA intermittent, we found during testing that the voltage tripler is capable of supplying better than 50 mA at 25 volts, which was more than enough to quickly smoke the 1/2 watt resistor we were using for a load! With a heavier resistor we were able to continue this test for more than 24 hours with no damage to the circuitry.

ADJUSTING THE HIGH VOLTAGE

As mentioned earlier, the high voltage Vpp is preset at the factory by the trimming resistors TL and TH. Under normal circumstances this adjustment need not be altered, but if it becomes necessary to change any of the voltage divider resistors (or possibly IC3) Vpp may be affected. Here, then, is the procedure for selecting TL and TH.

TL must be adjusted first. Connect pins 1 and 16 of the personality module socket together; this can easily be accomplished by inserting a module for an HMOS™ EPROM (such as the 2732A) in the socket.

Run the QED program and set it for one of the 16K byte EPROMs (2528 or 27128). Select the program function and tell QED you wish to program the entire buffer. When you hit C for continue Vpp will be turned on. If programming is completed before you are finished, simply repeat the procedure. DO NOT PLACE AN EPROM IN THE SOCKET!

Now measure Vpp at pin 12 of the personality module socket; it should be at 21 volts, plus or minus one half volt. If not, remove TL and begin substituting new values. A lower value for TL reduces Vpp; a higher value increases it. Note that Vpp will always be above 21 volts with TL out of the circuit.

Now remove the jumper between pins 1 and 16 of the personality module socket. Vpp should rise to 25 volts, plus or minus one volt. If not, remove TH and begin substituting as above. Vpp will always be above 25 volts with TH out of the circuit.

THE QED DRIVER PROGRAM

QED is the software which makes the EPRAM system "go". By placing the responsibility for most of the timing and control functions with the software, we were able to eliminate several latches and one shots from the EPRAM board, making the hardware simpler and less expensive. The price paid for this simplification is determining the DELAY constant when initially assembling the software.

QED is relatively large for its function (about 6K bytes), but a significant part of this bulk is devoted to ASCII text. If your application requires QED to fit in a smaller area, this would be the best place to begin trimming.

In an attempt to make QED as system independent as possible, the program contains its own terminal I/O routines. In fact, QED is so transportable that it can be installed, without modification, on virtually any 6800 or 6809 system using an ACIA based terminal and containing sufficient user memory. The standard user defined variables are the only setup required. Users of non-ACIA based I/O generally need only patch in a few vectors to existing routines, although special attention must be given to controlling character echo.

This manual will not attempt to explain every detail of the QED program (which is the function of the program's comments), but we will expand on a few items which need more detail. Between this manual and the program source code you should be able to glean enough information to make any necessary changes or modifications. (By the way, if you haven't tried to figure out what QED stands for yet, don't; it will only make you crazy. And don't ask us. We won't tell.)

PROGRAM ORGANIZATION

In the 6800 version of QED, the first six bytes are occupied by the traditional COLDSTART AND WARMSTART JUMP VECTORS. In the 6809 version, these are long branches and the first eight bytes are used. Since the current versions of QED do not have a true warmstart address, both vectors point to the top of the command loop, named WARMST.

Immediately after these vectors are a group of FCBs, FDBs, and EQUs which make up the USER DEFINED VALUES. These are the constants which must be set up to adapt QED to any given system.

Following this block is a second block of SYSTEM EQUATES, followed by a block of SYSTEM VARIABLES. Except for some tabular data in the next section these two blocks contain all the equates and variables for the entire QED system.

Next is a large block of TEXT STRINGS. In addition to user messages this section includes several menus and data tables. These special function strings and equates are located at the end of the text string block.

Now comes the juicy stuff! The first segment of QED program is the main

COMMAND LOOP, beginning with the label WARMST. The command loop has the responsibility of vectoring control to the various subprograms in response to menu entries.

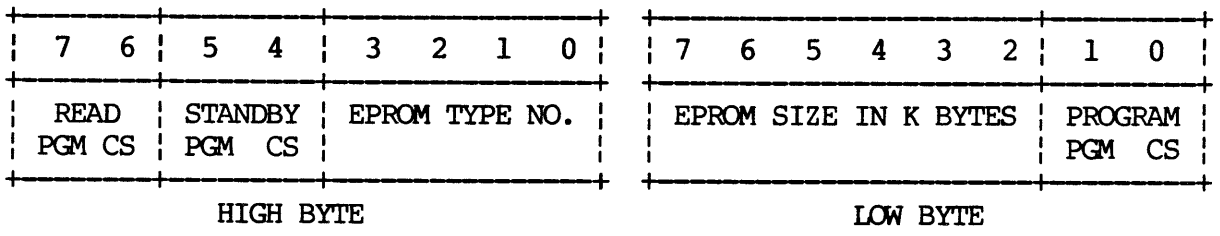
Following the command loop are a group of what we call "SUBPROGRAMS". These program segments are not subroutines, but are "called" to perform a specific function much like a subroutine. With the exception of the exit subprograms, all return control to the command loop at the WARMST point.

After the subprograms are a block of SUBROUTINES. The general program flow is for the command loop to vector to one of the subprograms, which in turn may call many subroutines to complete its task. Note that while subroutines call other subroutines, a subprogram is never "called" by another subprogram (although a few vector control, never to return).

THE OPT2 TABLE AND EPROM VARIABLE

The EPROM type to be operated on is controlled by a constant loaded into the double byte variable named "EPROM". This variable is initially set for the default EPROM type with an FDB during assembly, and is altered using the SELECT EPROM TYPE command.

This double byte variable performs several functions, and is fragmented as shown:



Taking the simpler portions first, EPROM SIZE IN K BYTES is a one of six code representing the EPROM size from 1K to 32K bytes. Thus it can be seen that, in software at least, the EPRAM system is ready for the 32K byte EPROMs (27256???). However, this will require a multiplexing of pin functions on the EPRAM board, and it remains to be seen if this can be done without hardware modification. This value is used by subroutine CBFEND to calculate the ending location of the data buffer, given the buffer starting address and the EPROM size.

EPROM TYPE NO. is a four bit binary number representing the type-code assigned to the EPROM. The ASCII letter is obtained by adding $40 to the type number; thus number 3 is $43 or EPROM type "C". EPRAM can access fifteen distinct EPROM types from its menu (although number 0 could be used it is not implemented), and therefore has room left for several new chips. This value is used by subroutine DISPHD to index into the ETYPE table and obtain the IC name for the menu headers.

The READ, STANDBY and PROGRAM bit pairs represent the values of the PGM and CS bits during the respective states. The READ state is the normal condition for the EPROM. When programming is to be performed, the EPROM enters the STANDBY state; this condition also exists between programming pulses. The PROGRAM mode is the time during which a memory location is actually programmed. (Note that there is no power-down or idle state implemented. This is not needed, since the Vcc to the socket is turned off when the EPROM is not being accessed.) These bit pairs are used by the SETCTL subroutine to build the READ, PGM, and STDBY masks which are merged with chip addresses when sending information to the EPROM.

ADDING NEW EPROM TYPES

Although it is possible to add a new EPROM to the QED menu, it is not particularly easy. This should not be a problem however, since adding a new chip won't be required very often! Here is an outline of the procedure if it becomes necessary:

First, modify the EPROM selection menu MENU2 to include the new IC. This is a simple matter of selecting a type-code and adding the required ASCII text.

Second, use the manufacturer's literature to determine the EPROM size and required bit pairs for READ, STANDBY, and PROGRAM. Build the double byte "EPROM" constant from these numbers and the EPROM TYPE number which corresponds to the code you have selected for the menu.

Third, add your new type-code letter and the "EPROM" constant to the OPT2 table.

Fourth, add the type-code letter and chip name to the ETYPE table in the SAME RELATIVE POSITION as your addition to the OPT2 table.

Of course to really do this job right you would re-letter all the menu entries so the EPROMs are still in the proper order, but this would be a monumental task! We therefore make this suggestion: if you find a new EPROM you can program (not another manufacturer's 2500 or 2700 series, but a truly different chip), send us all the information including the EPROM constant, and we will revise the QED program to accomodate the chip (and of course send you a copy).

And a P.S.: Don't forget that setting QED up for a new EPROM is only half the job. You must also create a personality module for the new chip!

Random thought: Has it occurred to you that, with the advent of the word processor, the true P.S. is for the most part extinct? It serves no useful purpose, since you can easily move back into the text of a letter and insert the renegade thought at the appropriate place. And, if you are sitting there reading this wondering what it has to do with anything, take heart; I am sitting here typing it and wondering exactly the same thing.

QED PROGRAM FUNCTIONS

In this section we will list the functions of the major QED subprograms and subroutines. A few explainations will be included, but the source code for QED should be consulted for detailed information.

WARMST This is the entry point for the main command loop. All subprograms except for exit functions return here.

KONST This subprogram is called by entering a K from the main menu. KONST fills a block of memory with a constant value.

CHANGE This subprogram is called by entering a C from the main menu. CHANGE allows you to examine and change the data in individual memory locations.

LDMEM This subprogram is called by entering an M from the main menu. LDMEM moves a block of data from one area of memory to another. This routine automatically determines whether the destination block is higher or lower than the source block, and copies data in the appropriate manner so that no information is lost in case of overlap.

LDPROM This subprogram is called by entering an L from the main menu. LDPROM copies part or all of the data in the EPROM to the buffer area.

RELOC This subprogram is called by entering an R from the main menu. RELOC moves the QED program to another location in user memory. The buffer area is not disturbed or moved. To perform this task the routine sets up the necessary pointers and limits, then copies the core of the LDMEM subprogram (SOMOVE through EOMOVE inclusive) to the TCA where it cannot be overlaid. RELOC then determines where it must vector after QED is relocated, and pushes this address onto the system stack as a pseudo return address before executing the move.

XEQCMD This subprogram is called by entering a D from the main menu. XEQCMD allows you to execute any of your OS commands which do not load into or use the area of memory in which QED resides. XEQCMD obtains the OS warmstart vector and replaces it with a pointer to itself before calling the OS. When the routine being used attempts to warmstart the OS, control is instead returned to QED, which replaces the original OS vector and executes a cleanup call.

ALTER This subprogram is called by entering an A from the main menu. ALTER is used to move the buffer to a new area of memory.

HDUMP This subprogram is called by entering an H from the main menu. HDUMP is used to generate a standard paged HEX/ASCII dump of any section of memory.

VERIFY This subprogram is called by entering a V from the main menu. VERIFY compares the contents of the EPROM, or a portion thereof, with the contents of a corresponding section of the buffer. Any mismatches are reported.

ERASED This subprogram is called by entering an E from the main menu. ERASED is used to verify that any portion of an EPROM contains only $FFs, which is the erased state of an EPROM. This routine merely sets a flag before vectoring to VERIFY.

SELECT This subprogram is called by entering an S from the main menu. SELECT is used to alter the value in the "EPROM" variable, thus setting QED for a new EPROM type.

PROGRM This subprogram is called by entering a P from the main menu. PROGRM is called to actually program an EPROM, or a portion of one, with the contents of the buffer.

EXMON This subprogram is called by entering an X from the main menu. EXMON clears the screen and exits to the system monitor.

EXDOS This subprogram is called by entering a Q from the main menu. EXDOS clears the screen and exits to the resident OS.

CLRSCN This subroutine homes the cursor and clears the screen.

MIDDLE This subroutine clears the screen and moves the cursor to the middle of the screen vertically.

DISPHD This subroutine displays the text pointed to by X, followed by the remainder of the menu header text. The "EPROM" variable is used to obtain the EPROM type-code and name for display, and subroutine DADDR is used to display the buffer limits. CLRSCN must be called before this routine.

DADDR This subroutine adds the size of the EPROM given by "EPROM" to the start of the buffer given in BUFST and places the results in BUFEND. The buffer limits are then displayed on the terminal.

SETCTL This subroutine builds the READ, STDBY, and PGM byte masks from the bit pairs given in the "EPROM" variable.

KEYERR This subroutine signals a keystroke error by backspacing the cursor over the error and ringing the terminal bell.

OUT4HN This subroutine displays the hex address pointed to by X on the terminal.

OUTHAA This subroutine displays the hex value in A on the terminal.

GETLIM This subroutine displays the text string pointed to by X, shows the current buffer limits, then obtains the starting and ending limits for

an operation. These limits are placed in BEGIN and LIMIT. If a "W" is entered for the beginning limit the buffer values are automatically inserted.

GETBEG This is the first half of the GELIM subroutine; it does everything except ask for an ending value.

LIMCHK This subroutine looks for an overlap of the operation limits and QED program. If one is found a warning message is displayed. LIMCHK is called only by mass change operations.

CORABO This subroutine displays the "C)ontinue or A)bort?" prompt, then obtains a response from the terminal. If an A is entered, the carry bit is set to show an abort was requested; the carry is cleared for a C. Any other character is rejected.

BADDR This subroutine inputs four valid hexadecimal ASCII characters from the terminal and builds a hex address from them in X. Non-hex characters are rejected, and the routine will not return control until four characters are entered.

CHKHEX This subroutine determines if the character in A is a valid hex digit, and converts it to its binary equivalent if so. The carry is cleared if the digit was valid and set if not.

CBFEND This subroutine adds the EPROM size indicated by the "EPROM" variable to the start of the buffer in BUFST and places the resulting end of buffer in BUFEND.

SAFETY This subroutine turns the low and high voltages to the EPROM socket off.

INITAD This subroutine initializes the address side of the PIA. It exits through LVOFF.

DATIN These subroutines control the direction of the data side of the PIA.
DATOUT DATIN sets the PIA for input, while DATOUT sets it for output. Both exit through HVOFF.

LVON These subroutines control the voltages to the EPROM socket. LVON
LVOFF enables the Vcc to the socket, while LVOFF disables it. In the same
HVON manner HVON enables the Vpp to the socket, and HVOFF turns it off.
HVOFF Several other subroutines exit through these functions.

LOADHI These subroutines are used to load data into the "upper" and "lower"
LOADLO data latches without disturbing the EPROM voltage lines. LOADHI momentarily pulses the CB2 line in the opposite direction of it's existing state; LOADLO serves the same function with the CA2 line.

WAIT50 These subroutines generate the required time delays for programming an
WAIT2X EPROM. WAIT50 returns after 50 milliseconds; WAIT2X calls WAIT50 twice for a one tenth second delay.

VGETCH This subroutine obtains a single character from the terminal. Only displayable characters are accepted, and lower case characters are converted to upper case automatically. The character is echoed back to the terminal.

ACIAIN This subroutine obtains a single raw character from the terminal. No echo is provided. This subroutine must be modified for non-ACIA based systems.

OUTST This subroutine sends the ASCII string starting at X to the terminal. A null indicates end of string.

OUTEE This subroutine transmits the character in A to the terminal. This subroutine must be modified for non-ACIA based systems.

LOOKUP This subroutine looks up an entry in a table. On entry, the code to match is in A with X pointing at the table. LOOKUP returns with the two bytes following the table code in the X register. If the code is not found the carry bit is set. A null indicates end of table.

OH, NO!

Let's hope you are reading this out of curiosity, and not because you need help. This section will run down some common problems and their most likely causes. We will not attempt to repair defective hardware or software bugs, but rather present possible installation or configuration errors and their solutions.

SYMPTOM: LV LED DOES NOT ILLUMINATE ON SYSTEM POWER-UP OR RESET

HARDWARE: The LV LED lights because the associated PIA control line becomes an input whenever the PIA is reset. The line floats high, turning the LV LED (and the socket low voltage) on. If this LED does not illuminate, make sure the EPRAM card is getting its unregulated 5V source (usually around +8 to +12 volts).

If this LED lights during normal operation and the programmer operates properly in all other respects, it may be that the floating line simply does not turn the LV switching circuit on. This is a question of component tolerance and, although unlikely, is certainly possible. If this is indeed the case it is nothing to be concerned about.

SYMPTOM: LV LED DOES NOT EXTINGUISH WHEN QED IS RUN

SOFTWARE: The most likely cause of this defect is that the incorrect address was supplied for the constant "EPRAM" when the program was assembled. Check this equate and correct it if necessary.

SYMPTOM: LV LED EXTINGUISHES WHEN QED IS RUN BUT SCREEN REMAINS BLANK

SOFTWARE: Most probably an incorrect address was supplied for the constant "ACIA" when the program was assembled. If your system uses a PIA for I/O or you have a video board, check your interface software for problems. Remember, QED wants to talk DIRECTLY to a standard ACIA. It does NOT use any of your OS or monitor I/O routines.

SYMPTOM: SCREEN DATA SCROLLS, GARBLED DISPLAYS, OR A TACKY SCREEN FORMAT

SOFTWARE: Check the values you supplied for the HOME CURSOR / CLEAR SCREEN sequence, as well as the backspace function. All of QED's displays were well thought out (at least WE like to think so!) and should be neatly centered and legible. The only displays which should scroll data off of the screen are the C command and whatever OS functions you you might call using the D command. All other displays should be presented in a paged format.

HARDWARE: Remember that QED was written for a standard 80 x 24 terminal format. If your screen format is significantly different you will have to restructure the QED text to prevent wraparound or miscentered screens.

SYMPTOM: SCREEN DATA LOOKS OK BUT (SOME) INPUT NOT ACCEPTED

SOFTWARE: This is almost impossible unless you are running a non-ACIA based system and supplied your own routines. With an ACIA, if you got the base address correct and are getting output then the input will automatically be OK. With non-ACIA systems, be sure of your I/O routines and vector addresses. If QED reacts to all inputs but some are rejected when they should be accepted, make sure you are stripping any parity bits from your input device.

SYMPTOM: INPUT CHARACTERS ARE DOUBLE ECHOED

SOFTWARE: This is another problem usually caused by a non-standard QED adaptation. Make sure the input routine you substituted for INEEE does not echo characters at any time. QED handles its own echo automatically. This bug is also evidenced by rejected characters making the cursor walk across the screen.

HARDWARE: If you are using a standard ACIA-based system, check to make sure your terminal is not echoing its own characters. Some terminal manufacturers call this a HALF-DUPLEX mode; it is not often used.

SYMPTOM: EVERYTHING LOOKS OK BUT EPROMS FAIL TO PROGRAM

SOFTWARE: If this is your first attempt at programming since installing QED, make sure you specified the correct "DELAY" constant. If this value is too small the EPROM will not be programmed; too large a value can damage the chip. Also be sure you have selected the proper EPROM type for the chip you are attempting to program. Is the chip erased? Remember, programming an EPROM means turning ones into zeroes; you can't go the other way!

HARDWARE: Make sure the correct personality module is installed for the chip you are programming. Also be sure the EPROM is inserted in the socket properly; with 24 pin chips there are five ways to do it wrong and only one way to do it right. This also applies to the personality module (but there is only one way to get it wrong!) Check the programming voltage, Vpp, to be sure it is at either 21 or 25 volts. A lack of high voltage could be caused by an excessively low unregulated 15 volt line (usually will run from better than 15 volts to 30 volts). Remember, The HV LED will illuminate during a programming attempt EVEN IF NO HV IS PRESENT. Also suspect a bad EPROM.

SYMPTOM: DATA READ FROM OR PROGRAMMED INTO EPROM IS REPEATED

HARDWARE: This symptom can be seen in several ways, such as each byte being repeated twice, or data repeating in blocks of eight bytes. It will evidence itself either when data is read from an EPROM into the buffer, or when an EPROM is programmed. It is usually caused by one of the EPROM's address pins not making contact with the socket. Keep dust and dirt out of the ZIF socket to help prevent this. This symptom can also be caused by a bad connection in the personality module socket, but in this case will only be seen as very large repeating blocks. Also suspect the EPROM.

SYMPTOM: ONE BIT IN EACH BYTE STUCK DURING READ OR PROGRAM

HARDWARE: This symptom is caused by an open pin connection, much like the above problem. An open data line will cause that line to float high, resulting in a stuck "1" during either a read or program operation. A stuck "0" is much less common. As above, keep the ZIF socket free of dirt and dust if this problem occurs. Remember that this symptom can also be caused by a bad EPROM.

ABOUT SURPLUS EPROMS...

Several of the problems above mention the possibility of a bad EPROM. OF ALL THE EPROMS WE HAVE EVER PROGRAMMED, WE HAVE NEVER FOUND A BAD IC. The majority of manufacturers have excellent quality control and catch most problems before an IC leaves the factory.

Surplus chips are another matter. Many companies buy up "culls" from the manufacturers; these are chips which FAILED the QC tests. Most dealers are not unscrupulous enough to sell out-and-out bad parts (although some don't bother to test), but many will be out of spec. We know of one customer who has a bunch of EPROMs which blow up if programmed with a 50 Msec pulse; he has to use a 30 Msec pulse on these. We have used chips from the same manufacturer before and have had no problems; the specs clearly state a 50 Msec pulse is required to program them. This person goes to hamfests often, and we believe he bought a bunch of culls, although he blames his problems on the manufacturer.

This is not intended to frighten you away from hamfests and surplus dealers; some excellent buys can be obtained this way. We simply want to put our remarks about suspecting bad EPROMs in the correct light: if you buy only from semiconductor houses and first-quality distributors, be less concerned about the chip being bad when looking for problems.

ALL EPROMS ARE NOT CREATED EQUAL

EPROMs range in size from the 1K 2508 and 2758 up to the mammoth 16K 2528 and 27128. (Can there be any doubt that a 32K byte "2556" or "27256" is on the way?) Every time the amount of memory in an EPROM is doubled, one more address line is required to access the added memory.

When the lowly 2708 was designed, no one could think about the higher density chips, so increasing memory meant moving pin functions around to make room for the new address lines. This got totally out of hand once the density hit 8K bytes, and manufacturers had to add four new pins. This is one of the contributors to the pinout melee in the EPROM industry. Another factor is secrecy, lack of communication, and just plain bull-headedness. ("Sure, we know THEY are working on a 4K byte EPROM too, but we're not about to conform to THEIR pinout; let THEM conform to OURS!")

Fortunately, the situation is not as bad as it could be. Thanks in a large part to the work of JEDEC (the Joint Electron Device Engineering Council), we have an industry standard EPROM pinout which fixes the position of most of the pins.

WHAT DOES THE PERSONALITY MODULE DO?

EPRAM's personality module is basically a jumper assembly designed to route the address and chip control signals to the remaining "funky" pins. In addition, the module sets the programming voltage and provides transient suppression where needed.

If you will examine the EPROM pinout chart in the appendix, you will see that only pins 18 through 21 change function on a 24 pin EPROM, and that pins 2*, 24, and 27* vary on 28 pin chips. (See the paragraphs on hardware in the ADVANCED INFORMATION section for data on our EPROM numbering system.) What we have done is brought some of these socket pins out to the personality module socket, along with the associated address and control lines, so they may be jumpered together as required.

Now look at the pinout of the personality module socket, also in the appendix. You will notice that EPROM pin 19 is not brought to the socket. This is because pin 19 is address line A10 in all chips except for the 1K byte versions, where this line is called "Array Select" or something equivalent. Since this pin always performs the same function in the chips EPRAM handles, it is not brought to the module socket. (See the section ALL ABOUT EPROMS for more information on line A10.) What ARE brought to the socket are the address lines A11 through A13, the two control lines called CS and PGM, 5 volts (Vcc), 21/25 volts (Vpp), ground, the Vpp select line, and EPROM socket pins 2*, 18, 20, 21 ,24, and 27*.

Building a personality module is a simple matter of connecting the right signals to the right EPROM pins. The appendix contains diagrams of the internal connections for all the modules needed by EPRAM; you have simply to look up the required EPROM and wire a DIP header accordingly.

DESIGNING NEW PERSONALITY MODULES

We will now examine the creation of a personality module, in case you must build a module which is not listed here, or are just plain interested. We will illustrate our discussion with the design of a module for the 2532. All pin connections which are not given in parentheses are for the personality module.

For most EPROMs designing a module is a relatively simple process. First, connect the required address lines to their appropriate pins. For a 2532 we need twelve address lines. A0 through A10 are already connected to the socket, so we need only jumper pin 14 (A11) to pin 7 (EPROM pin 18).

Next you must consider Vcc. All 24 pin EPROMs have their Vcc applied to pin 24, so you should jumper module pin 13 (switched 5 volts) to pin 4 (EPROM pin 24). Note that 28 pin chips receive their power through pin 28*, which is permanently connected to Vcc.

Now comes Vpp. You must jumper EPRAM's HV source to the appropriate pin on the EPROM; it is usually labeled Vpp, or Vpp/something. In this case we would connect module pin 12 (switched Vpp) to pin 5 (EPROM pin 21).

The next consideration is whether the Vpp pin should be held to 5 volts or ground when reading from the EPROM. Most chips require it to be held to Vcc; if this is the case you should connect a small diode (a 1N914 or 1N4148 is acceptable) between the module's Vcc pin 13 and Vpp pin 12, with the cathode to pin 12. This will hold the EPROM's Vpp pin to 4.4 volts during read (a very legitimate logic one), while preventing the high voltage from feeding back into the Vcc line. A very few EPROMs require Vpp to go to ground to read data; for these simply leave the diode out.

Now connect the PGM line. This is the line which pulses for 50 milliseconds to actually program a memory location. It is to be connected to the EPROM's PGM pin, which can also be called PGM/something or just something. If it's not listed on the pinout diagram, look elsewhere in the manufacturer's data. In our case we connect module pin 10 (PGM) to pin 6 (PD/PGM).

Next you must consider the CS line. This pin is connected to the second chip control line, which may be called Output Enable or Chip Select. This is the enable or select line which does NOT share the program function. For a 2532 module pin 11 is ignored. In the case of multiple control pins, they can all be tied together (assuming the polarities are identical) or the extra pins can be connected to pin 13 (Vcc) or pin 16 (ground) as required to enable them.

Most EPROMs require a 25 volt Vpp, which is normally supplied by EPRAM. EPROMs which are built using HMOS™, however, would be permanently damaged by a 25 volt Vpp; they require 21 volts instead. This lower voltage is obtained by strapping module pin 1 (21VSEL) to pin 16 (ground). Since the 2532 uses a 25 volt Vpp this strap is also ignored.

Finally, you must consider any requirements particular to the specific EPROM which have yet to be covered. For example, chips using INTEL's HMOS™ process or equivalent usually require a .1 microfarad capacitor between pin 12 (Vpp) and pin 16 (ground) to prevent transients from damaging the EPROM. The 2532 has no such requirements.

WARNING: Some manufacturers like to bury this information. For example, INTEL clearly states in bold type that when applying Vpp EXCEEDING 22V WILL DAMAGE THE 2732A. It's hard to miss. However, the warning about transients is buried in the third paragraph under PROGRAMMING and is stated rather casually. Be sure to read all the fine print, and double check your wiring before trying out a new module.

ALL ABOUT EPROMS?

Well, only in the sense that this section is all about EPROMs; we certainly couldn't tell you everything there is to know! This portion of the manual runs down some basic information that we felt might be of use to you.

STATIC ELECTRICITY DAMAGE

Yes, yes, you've heard this before. Over and over. But it doesn't hurt to mention it again. MOS BASED ICS ARE SUSCEPTIBLE TO DAMAGE FROM STATIC ELECTRICITY! USE EXTREME CARE WHEN HANDLING THEM!

Don't fondle your EPROMs. Leave them in their anti-static carriers until you are ready to program or use them, then move them immediately to their sockets.

Be especially careful during the winter. Static conditions are at their peak during cold (and dry) weather. Remember all those times you were bitten sliding out of the car last winter? Think what that would do to an EPROM. (And you guys in Florida take pity on the rest of us.)

Here's a new one you may not have heard: HITACHI (a well-respected manufacturer of EPROMs) says that a static charge can be induced in the surface of an EPROM's window, which can damage the chip. This means that you can destroy an EPROM even though it may still be in its carrier. Be careful. Avoid rubbing the window with nylon fabric, plastic film, etc.

Support the back of the EPRAM PC board with your fingertips while inserting or removing an EPROM; let your fingers touch the PC traces. This not only prevents the board from flexing on the buss connectors, it allows any static potential in your body to drain away before inserting the EPROM in the socket.

ERASING EPROMS

An EPROM is erased by exposing it to "ultraviolet light". Ultraviolet is a term for those frequencies of electromagnetic radiation just above the visible range. Like visible light, ultraviolet spans a range of wavelengths. This range has been broken down into two categories, termed "shortwave ultraviolet" and "longwave ultraviolet".

Longwave ultraviolet is energy near a wavelength of 3660 Angstroms, and is the type of light normally used to excite "fluorescent" posters and glow-in-the-dark items. They can be found in tourist caves (to make the natural rock formations fluoresce), discos, and teenager's rooms. Longwave ultraviolet radiation is mostly harmless to the eyes.

Shortwave ultraviolet is that band of energy near 2537 Angstroms, and is used for many technical applications, such as sterilization and erasing of EPROMs. (EPROMS are sensitive to any radiation with a wavelength shorter than about

4000 Angstroms.) SHORTWAVE ULTRAVIOLET RADIATION IS HAZARDOUS TO YOUR EYES, AND CAN CAUSE PERMANENT DAMAGE! Never look at an EPROM eraser's lamp when it is on. Most erasers have sealed or filtered enclosures to prevent the escape of ultraviolet, and have interlocks to extinguish the lamp if the cover is removed. If you will be using a homebrew eraser you should use extra caution, as these safety features are usually ignored. Be especially careful of others who may be ignorant of the dangers, such as young children. WE DO NOT RECOMMEND HOMEBREW EPROM ERASERS UNLESS FULL SAFETY PRECAUTIONS ARE TAKEN.

The amount of time required to erase an EPROM depends on the amount of ultraviolet energy impinging on the surface of the silicon chip. This is determined by both the intensity of the source, and its distance from the IC. For example, a 12 mW/cm^2 source will erase an EPROM in about 20 minutes at a distance of about one inch from the EPROM window.

Commercial EPROM erasers are rated for normal erase time; you should not attempt to shortcut this exposure. An under-exposed EPROM may read as erased, but many of its cells may be in a borderline or unstable condition; a memory bit which is a one may not remain a one. It is usually not worth the potential hassle for the few minutes saved.

There is a certain amount of controversy regarding the use of EPROMs without a protective label once they are programmed. Although it is good practice to cover the quartz window once an EPROM is programmed, you should know the facts. Natural daylight contains a percentage of ultraviolet light. If left in direct sunlight during normal "earthly" conditions, an EPROM will be erased in approximately one week, although some bits will become "soft" in a shorter time. Normal fluorescent tubes also emit a certain amount of ultraviolet radiation; an uncovered EPROM would be erased after about three years. It can therefore be seen that, although the label should always be used for safety's sake, short exposures to normal lighting conditions will have little effect.

THE 2758 EPROM - NOBODY'S PERFECT

Regardless of the amount of control exercised during IC manufacturing there are bound to be a few bad chips. Some will be totally non-functional, a few wildly out of spec, maybe some that are "almost" perfect. This happens with all chips, and EPROMs are no exception.

Fortunately for memory manufacturers, a large percentage of the failures are one or two bits bad in the whole chip. (With 16,000 locations or so on a chip, the odds of one being bad are mighty high!) We said fortunately because if one bit is bad then the rest may be good. There is no law which says the IC cannot be sold as "half good", and this is exactly what is done in some cases; specifically, the 2758.

These chips are "half good" 2516 or 2716 EPROMs; you can prove it to yourself by looking at the pinout. Pin 19 is labeled AR, which stands for "array select" or "select reference", depending on which manufacturer's terms you prefer. If you purchase a straight 2758 from INTEL or a 2758-JL0 from TI this

pin must be grounded; you can also special order a 2758-S1865 (from INTEL) or a 2758-JL1 (TI) where AR must be tied high. Consider that this pin just happens to be the A10 pin on 2516 and 2716 chips, and that the A10 line selects which half of the EPROM you are accessing in a 2K chip, and it all becomes rather obvious.

(This is, by the way, nothing to be upset about. Many manufacturers use this technique; in fact, the 32K RAM chips used by Radio Shack in their Color Computer™ are failed 64K chips of which they use one half. If this sort of failure were simply discarded the result would be higher prices for everyone, and possibly the non-existence of some ICs, like single voltage 1K EPROMs.)

PROGRAMMING THE 2758

Now this situation brings up a minor problem; instead of having a simple 2758, we have two of them to consider: a "lower half" 2758 where pin 19 must be grounded, and an "upper half" 2758 for which pin 19 must be tied high. How do we program this (these) chips?

Remember that most 2758s you will encounter will probably be "lower half" versions, with AR held low. Recall also that, except for AR being A10 on 2K EPROMs, these chips are identical. We can therefore program a 2758 exactly as if it were a 2516 or a 2716, as long as we only program 1K of data; this is exactly what QED does when the 2758 is selected. This is, by the way, how EPRAM can use the same personality module for the 2508, 2758, 2516, and 2716. This then eliminates the "lower half" 2758s, leaving us with only the less common "upper half" versions to worry about.

Since the 1K and 2K chips are identical except for memory size (for our purposes), the easiest way to handle the "upper half" 2758 is to tell QED you are programming a 2516 or 2716 (it doesn't matter which), then use only the SECOND HALF of the buffer for programming. For example, if the buffer were situated from $8000 to $87FF, you would load your program into it from $8400 to $87FF (the "upper half"!) and program only this portion into the EPROM. Since QED will think it is programming a 2K EPROM, it will activate A10; because you are in the upper 1K this line will stay high and, mystically, magically, your renegade 2758 will be programmed!

(An alternative for an "upper half" 2758 would be to create a special personality module in which the AR pin is tied to Vcc. This approach would allow you to use QED's 2758 option without fuss, but would require swapping modules more often.)

CONVERTING TRIPLE VOLTAGE SOCKETS FOR SINGLE VOLTAGE EPROMS

There are bound to be a few EPRAM purchasers who have an old 2708 or two in their systems. These paragraphs outline a method for replacing triple voltage EPROMs with their single voltage equivalents. This will make the system totally compatible with the new chips, with the bonus of lower power requirements and less heat generation.

(As of this writing, triple voltage 2708s are rising slightly in price while single voltage chips continue to fall. You can now (November, 1982) purchase a single voltage 2716 for the same price as a 2708! It is our belief that as more and more systems move to single voltage chips and the older equipment is phased out, the price of triple voltage chips will rise to exorbitant levels. This sort of thing has happened before... have you tried to buy a vacuum tube lately?)

There are only three triple voltage EPROMs in the 2500/2700 series. The 2708 1K byte chip is by far the most popular. The 2704 is a 1/2 K byte chip which was never widely used. The triple voltage TI 2716 can be found in a few dedicated applications like terminals and printers, but not much elsewhere. We are therefore going to detail the replacement of a triple voltage 2708 with a single voltage 2508, 2758, 2516, or 2716. If necessary you can extend the method to the other triple voltage chips as required.

WARNING: This data is provided for your information only. We take no responsibility for any changes you make to your system, or any damages you may incur, because of this discussion. You should determine for yourself the feasibility of these changes relative to your application.

The first step in any conversion would be to program a new single voltage EPROM with the contents of the triple voltage version you are replacing. Since we are assuming that this chip is actually IN your system, this is a simple matter of placing the buffer over the existing EPROM and performing a program sequence. (If the IC is not in your system, and you have no capabilities for reading a triple voltage EPROM, the problem is of course very different.) If you are using a 2K EPROM to replace a 2708 because of cost considerations (they are for the most part cheaper than single voltage 1K chips), simply program the first half of the IC, leaving the second half untouched.

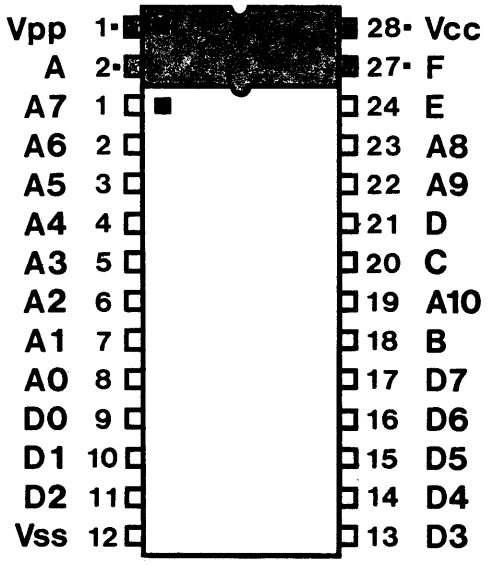
Once you have created a replacement EPROM, you can power down the system and remove the board for the required changes. If you will compare the pinouts of a 2708 and any of the single voltage 1K or 2K chips you will see they are very similar. In fact, only two to three cuts and jumpers are required for the conversion.

Begin by cutting the +12 volt source away from pin 19 of the existing socket. On the new chip this is AR or A10, and should be connected instead to ground (unless you are using an "upper half" IC, see above).

Next cut away the -5 volt bias source from pin 21 on the socket. On the new chip, this is the Vpp pin, and should be connected to Vcc (+5 volts or pin 24) for normal read operation.

This leaves pin 18, which was the PGM pin of the 2708, and is the CE-inverted of the new chip. On the original socket this pin was probably grounded, so you have a choice. If you leave it alone, the chip will remain enabled at all times, and its output buffers will be controlled by the old chip select line. This mode draws a considerable amount of Vcc current at all times; 57 to 100 milliamps as opposed to 6 to 10 milliamps on the old 2708. (Remember however that the 2708 required two other supplies and all together required from 85 to 120 milliamps, so the TOTAL current drain will still be less.) The other alternative is to cut pin 18 loose from ground and tie it instead to pin 20, which is now OE-inverted; in this mode both the output buffers and the chip select line are controlled. The current drain is still 57 to 100 milliamps DURING ACCESS, but only 10 to 25 milliamps in standby, which is far less than the 2708. Still, in dedicated applications with many EPROMs or a marginal five volt supply the current requirements must be carefully considered.

Of course the possible changes don't stop here. You can usually replace two 2708s with one 2516 simply by altering the board's select logic and connecting A10. The new higher density EPROMs open a world of possibilities.



This is an illustration of our EPROM pin numbering system. 24 pin EPROMs (represented by the unshaded portion) are numbered normally. 28 pin ICs are represented by adding four pins to the top of the package. These new pins are numbered as they would be on a 28 pin chip, but the pins held over from the 24 pin package retain their original numbers. To convert our numbers to standard 28 pin numbers, add two to pins 1 through 24.

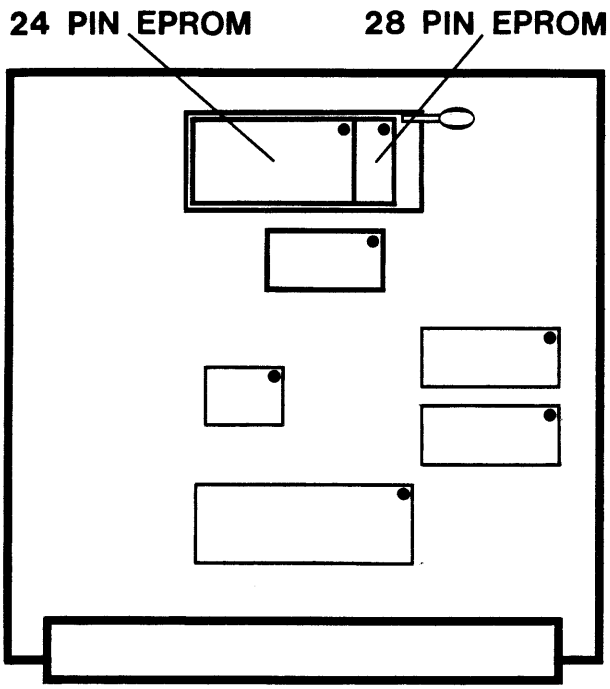
The pins marked A through F indicate those pins whose functions vary from one EPROM type to another. These are also the pins which are brought out to the personality module. The chart below shows the functions of these pins on all 2500 and 2700 series EPROMs, as well as some related ICs.

	EPRAM Compatible EPROMS											Other ICs of note				
	2508	2758	2516	2716	2532	2732	2732A	2564	2764	2528	27128	2708	2716	2816	6116	68764
A								$\overline{OE1}$	A12		A12					
B	$\overline{CE}$/PGM	$\overline{CE}$/PGM	$\overline{CE}$/PGM	$\overline{CE}$/PGM	A11	$\overline{CE}$/$\overline{PGM}$	$\overline{CE}$/$\overline{PGM}$	A11	$\overline{CE}$		$\overline{CE}$	PGM	$\overline{CE}$/PGM	$\overline{CE}$/PGM	$\overline{CE}$	A11
C	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{CE}$/$\overline{PGM}$	OE/VPP	OE/VPP	$\overline{CE}$/$\overline{PGM}$	$\overline{OE}$		$\overline{OE}$	$\overline{CE}$	A10	$\overline{OE}$	$\overline{OE}$	A10
D	VPP	VPP	VPP	VPP	VPP	A11	A11	A12	A11		A11	VBB	VBB	VPP	R/$\overline{W}$	CE/VPP
E	VCC	VCC	VCC	VCC	VCC	VCC	VCC	VCC	NC		A13	VCC	VCC	VCC	VCC	A12
F								$\overline{OE2}$	$\overline{PGM}$		$\overline{PGM}$					

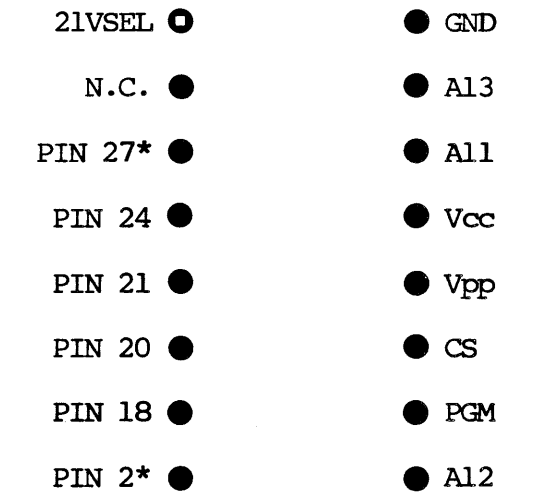
NOTES:

On triple voltage EPROMs pin 19 is Vdd. On 1K EPROMs pin 19 is AR or NC.
OE (Output Enable) is a control line for the output tristates.
CE (Chip Enable) is a combination device select and OE.

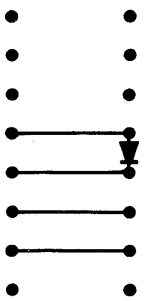
Some manufacturers number data pins from 1 to 8 rather than 0 to 7.
TI's PD (Power Down) is equivalent to INTEL's NOT CE (Chip Enable).
TI'S NOT CS (Chip Select) is equivalent to INTEL's NOT OE (Output Enable).



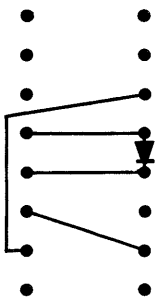
IC ORIENTATION



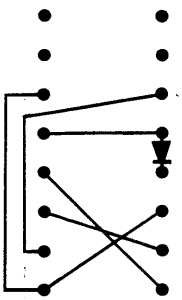
Personality Module Pin Functions



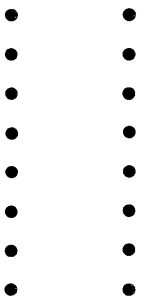
2508
2758
2516
2716



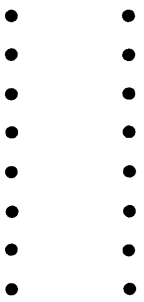
2532

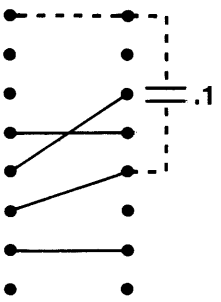


2564

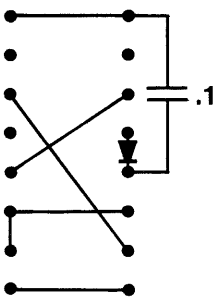


2528

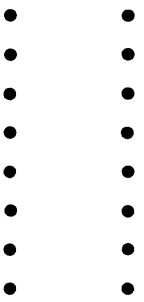




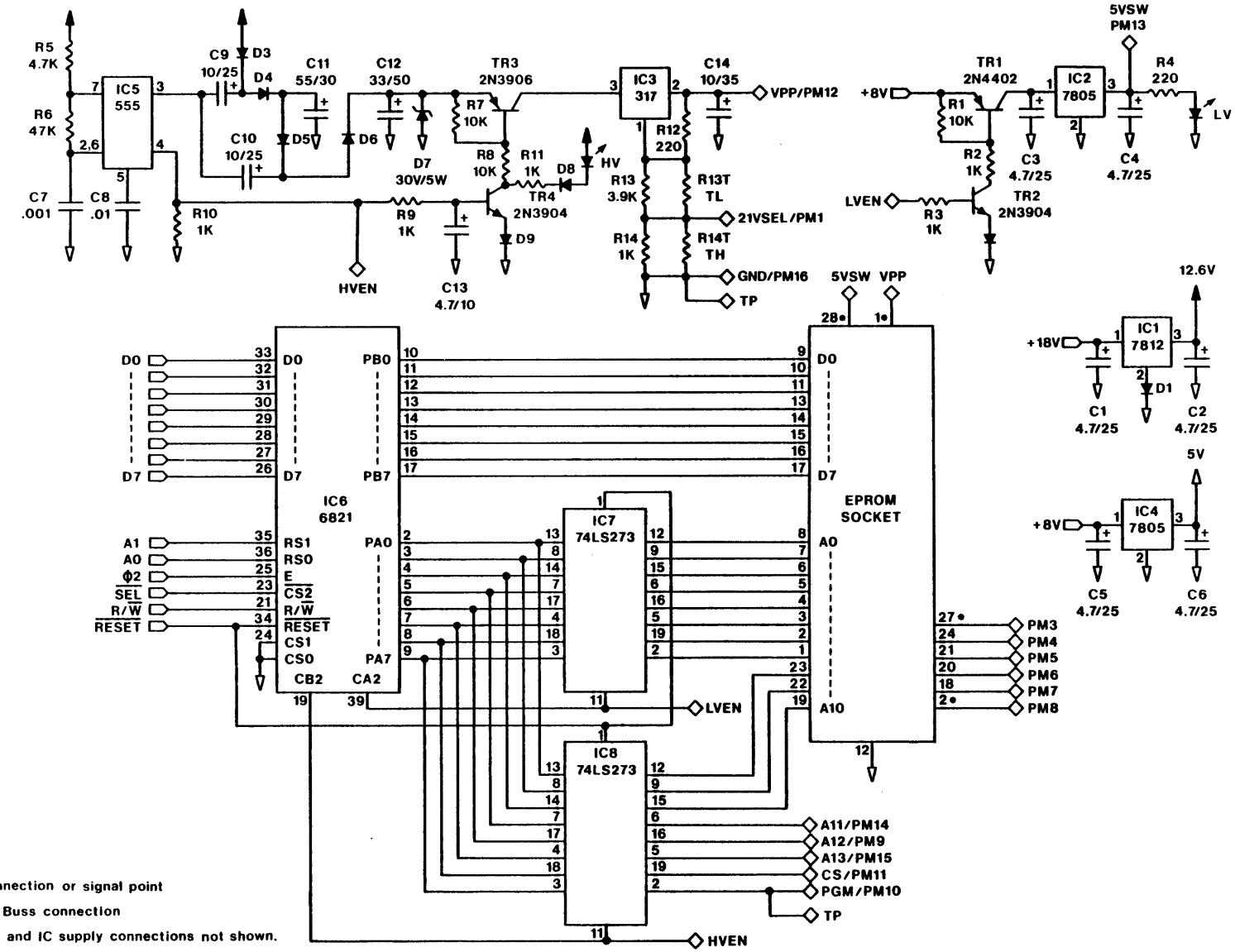
2732
(Add dotted
for 2732A)



2764



27128



◇ = Interconnection or signal point

▭ = System Buss connection

Bypass caps and IC supply connections not shown.

All diodes 1N914 except as noted.

*All EPROM socket numbers refer to 24 pin numbering (ignoring pins 1, 2, 27, 28) except as noted by asterisk.

DWN RHR	UNIQUE TECHNOLOGIES	
CHK	NAME: EPRAM50 (V1R0)	
APP	SHEET: 1	DWG ID: EPRAM50-8208.270 0
DATE 8/27/82		

4000 X 250 = 1000,000

0100 - 1718	QED2716.440
D372 - - - D3E6	SWTPCIO.OVR
9D00 - 9F26	ANDY-M2.440

2716	8000 - 87FF	E000
2732	8000 - 8FFF	E000
2764	6000 - 7FFF	C000

FA

A000
C000
16000

C000
2000