EPROMMER 64

instruction manual

DATEL UTILITIES

BEFORE CONNECTING OR DISCONNECTING EPROMMER 64 ALWAYS MAKE SURE THAT YOUR COMPUTER IS SWITCHED OFF. FAILURE TO DO SO WILL RESULT IN SERIOUS DAMAGE.

Epromer 64 is plugged into the user port of the 64\128. The software when loaded will display a control menu.
The various sections are connected with the different aspects of eprom porgramming i.e. reading, programming, verifying etc.
The command section is as follows:

T Selects eprom type. Pressing this key will scan through the various chip sizes from 2716-27256.

P Selects programming speed. Pressing ths key will scan through 3\5\50ms. The best speed to select is 5ms though 3ms can be used for faster burn. However 3ms may give unreliable results on some brands. It should be noted that old 2716 + 2732 devices can only be programmed at 50ms. For this reason 50ms is automatically set when these chips are selected.

C Selects programming voltage. Pressing this key will scan through the various voltages 12.5\21\25 volts. When you select chip type the voltage will set to the most popular for this device. Always check that your device is suitable for this setting and change accordingly.

* Selects 1\0 status. Pressing this key will toggle between RAM\ROM. The purpose of this setting is to select whether to read from the ROM or from the RAM under the ROM.

Q Selects program range. Pressing this key will toggle between all of the chip or a range/ section of the device. If range is selected then the parameters for the range should be set (see below).

$ Displays disk directory.
L Loads a file from disk.
@ Send a disk command.
S Save a file to disk.
M Enter monitor (See mon commands).
X Exit program.
F Fills memory with one byte.
B Checks for blank eprom.
E Will erase an E.Eprom.
V Verify an eprom against ram.
R Read an eprom into ram.
W Write an eprom.

A Allows you to set the range. The current range is displayed at the bottom left of the screen. Pressing A will allow you to change these parameters. The cursor will flash alongside the current setting and you can enter the new value then RETURN. Pressing JUST RETURN will accept the current value. The various values are as follows:

DATEL ELECTRONICS

RAMSTART	This is the start location in computer ram where the file you are going to program into an eprom begins.
EPROMSTART	This is the start location in the eprom to be programmed i.e. 0000 would program the chip from the start. 2000 would program from 8k upwards in the chip.
TASK LENGTH	This value sets how much of the chip is to be programmed. i.e. an eprom start address of 0000 and a task length of 2000 would program that chip in the first 8k of the device.
RAM END	This value will change when ramstart and task length are set.
EPROM END	This value will change when epromstart and task length are reset.

So you can see that any part or all of a chip can be programmed at one time. For instance if you wanted to program a replacement kernal chip for your 64 you would need a 16k chip to hold the old 8k system plus the new 8k system. An example of programming this task would look something like this:-

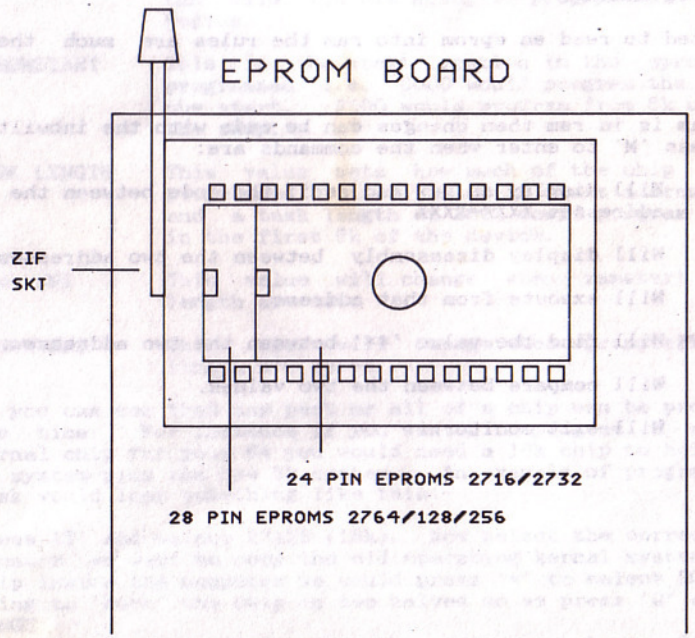
Press 'T' and select 27128 (16k). Now select the correct voltage. Because we want to copy the old operating kernal system from the chip inside the computer we would press '*' to select ROM. We are going to 'burn' the chip in two halves so we press 'Q' and select RANGE.
We will now press 'A' to alter the range parameters. Since we are going to copy the kernal rom we set RAMSTART to E000. . This means that we will extract the code from the kernal at E000 as the source for our programming. Now set EPROMSTART to 0000 because we are going to put the old kernal code into the bottom half of our new chip. Now we turn to TASK LENGTH. This we set to 2000 (8k). Thats it the parameters are set, we can now select 'W' and write the eprom. The screen will show the 'burn' as it progresses. Afterwards press 'V' to verify the contents of the new chip against the source (in our case the kernal). Next we move to the kernal code. We select 'L' and load our new file from disk. It will load into the RAM under the kernal or indeed any other free 8K space in ram. The eprom size and speed are unchanged but we press '*' to select 'RAM'. Next we move to 'A' to alter the parameters. If the new file has been loaded E000 then we set RAMSTART to E000. Remember that we could have loaded this file into any free 8K block and so a different RAMSTART value would be entered.
The value for EPROMSTART should now be changed to 2000. This is because we now want to program the top half of the 16K chip. TASK LENGTH is unchanged since our new file is also 8K.
Again we got to 'W' to write the chip 'V' to verify.

This is perhaps a complicated example but it does cover most aspects of programming.

If you wanted to read an eprom into ram the rules are much the same.

MONITOR:
When a program is in ram then changes can be made with the inbuilt monitor. Press 'M' to enter when the commands are:

M XXXX XXXX	Will display as hex and ascii the code between the two addresses XXXX-XXXX
D XXXX XXXX	Will display disassembly between the two addresses.
G XXXX	Will execute from that address.
H XXXX XXXX **	Will find the value '**' between the two addresses.
C XXXX XXXXX	Will compare between the two values.
X	Will exit monitor.



DATEL ELECTRONICS

CARTRIDGE DEVELOPMENT SYSTEM. INSTRUCTIONS.

The cartridge development system is a complete kit of parts to build an 8k or a 2x 8k switchable cartridge.
In its simplest form the board can be formatted as an 8k cartridge to appear in memory at $8000-9FFF, $A000-BFFF or even replace the kernal at $E000-FFFF.

By looking at the circuit diagram you will see that fitting the wire link will ground the EXROM line and hence the cartridge will appear at $8000. ROML is already connected to the chip enable. If in addition the PAD1 link is made then GAME will also be grounded and the cartridge will then appear at $A000 but this time enabled by ROMH (pad4). A second eprom could be incorporated using this method and a 16k cartridge be built.

The eprom provided is a 27128 16kdevice but our board really treats it as two 8k devices. When the board has been configured as a simple 8k cartridge (where ever in memory) the switch can be used to introduce either the bottom or top half of the chip. Thus two completely different 8k programs can be switched in at will.

We have tried to make the board as versatile as possible both for you and ourselves since we base most of our own products on this board. Further examination of the circuit diagram will show that many more methods of configuration are catered for. For instance both the Game and EXROM lines can be connected to the I/O1 and I/O2 lines. These lines are under software control and therefore a cartridge can be controlled by software.

The Epyx Fastload cartridge uses the I/O lines to make itself invisible to the system.

Please read the booklet carefully and all will be revealed. The above information is only an overview of the options available for this product. The rest of this booklet will deal with the various control lines to configure the system. Much of the information is not readilly available elsewhere, indeed Commodore themselves treat this subject as though it was some sort of secret.

When you have read the rest of the instructions you will appreciate that it not that complicated to achieve great things. On the other hand if you find it difficult just read through it again and we are sure you will pick it up. We have made it as concise as possible and it is an area that is well worth understanding.

Please be carefull when programming the Eprom and remember that if you are going to use it as two 8k s then set your programmer as such.

An attractive case is supplied to finish off your cartridge. This unit is ready drilled for the bank switch and reset button also provided.

If you come up with a good product that you may consider is worth putting into production then Datel will be pleased to quote for supply of cartridge kits in quantity.

Alternatively if it is really good why not send it for evaluation and we may be interested in marketing it for you.

HARDWARE CONTROL OF THE PLA

The memory that the microprocessor sees may also be controlled by hardware. Hardware control requires an actual connection from the pins on the cartridge port to ground. Two of the lines connected from the PLA to the cartridge port will control memory configuration. The PLA will monitor the voltage level of these two lines. These two lines are called the EXROM line and the GAME line. These two lines are normally high (+5v). When either (or both) of these lines are grounded the PLA will reconfigure the memory that the microprocessor sees.

Grounding only the EXROM line will cause the PLA to reconfigure memory so that the microprocessor will look to the cartridge port to find the memory from $8000-$9FFF. All of the other memory locations will remain intact. BASIC ROM, KERNAL ROM and the I/O devices will remain in effect. Under normal circumstances the EXROM line would be grounded only if a cartridge had been installed. If we were to ground the EXROM line without a cartridge installed the microprocessor would not find any memory at these locations ($8000-$9FFF). The PLA does not care if any memory exists at the memory locations that the microprocessor is looking at. If we ground the EXROM line without plugging in a cartridge, the PLA will prevent the microprocessor from seeing any memory other than what is at the cartridge port (nothing in this example). The microprocessor will only find random garbage in this area. This is a way for the PLA to prevent the microprocessor from seeing the RAM normally at $8000-$9FFF. REMEMBER THAT WHEN THE EXROM LINE IS GROUNDED THE PLA WILL CAUSE THE MICROPROCESSOR TO SEE ONLY THAT MEMORY THAT IS PLUGGED INTO THE CARTRIDGE PORT. THIS WILL OCCUR WHETHER THERE IS A CARTRIDGE PLUGGED IN OR NOT!

Grounding only the GAME line will cause the PLA to reconfigure memory so that the computer will be able to use cartridges designed for the "ULTIMAX" system. The KERNAL ROM and the BASIC ROM will be switched out and the microprocessor will look to the cartridge port for memory in the $8000-$9FFF and the $E000-$FFFF range. This configuration of memory will cause the microprocessor not to see ANY memory in the following areas of memory; $1000-$7FFF and $A000-$CFFF ('images' may appear in these open areas). Memory locations $0000-$0FFF will appear as the normal RAM and $D000-$DFFF will appear as the normal I/O devices. The microprocessor will look for memory locations $8000-$9FFF and $E000-$FFFF on the cartridge port. Again, this memory configuration is only for those cartridges that emulate the ULTIMAX system.

Grounding BOTH the EXROM and the GAME lines at the same time will cause the PLA to reconfigure memory so that the microprocessor will look to the cartridge port for memory at locations $8000-$BFFF. This configuration will allow the use of 16K of continuous cartridge memory. 8K will reside in the normal area of cartridge memory ($8000-$9FFF). The other 8K will reside in the area of memory that is normally reserved by BASIC ($A000-$BFFF).

This memory configuration will also allow for the programmer to switch between the RAM and ROM located at memory locations $8000-$9FFF. By controlling the LORAM line the programmer may select RAM or cartridge ROM. When the LORAM line is high the PLA will cause the microprocessor to see ROM at location $8000-$9FFF. When the LORAM line is low the PLA will cause the microprocessor to see RAM at locations $8000-$9FFF and the microprocessor will still see the cartridge ROM located at $A000-$BFFF. In other words, trying to turn off the BASIC ROM with LORAM when GAME and EXROM are both grounded **will** turn off the cartridge memory at $8000-9FFF but **will not** turn off cartridge memory at $A000-BFFF!

We have now covered the major functions of the PLA and microprocessor combination used in the C-64 as they relate to memory management. The PLA also has a few other important functions. When the microprocessor writes to an area of memory that contains both RAM and ROM (BASIC ROM $A000-$BFFF, for example) the PLA will allow the microprocessor to write to the underlying RAM. The PLA will decode the microprocessor's instructions when it is reading and writing. The PLA will then "decide" what memory that the microprocessor should have access to (RAM or ROM). If the microprocessor is going to write (STA) a value in memory, the PLA will select the appropriate memory (ROM cannot be written to). If the microprocessor will be reading (LDA) a value from memory, the PLA will select the proper area of memory based upon the LORAM, HIRAM, EXROM and GAME lines. The one deviation from the preceding example is when the microprocessor writes to the memory at $D000-$DFFF. This memory normally contains the I/O devices, rather than RAM or ROM. Because of this, the PLA will allow the microprocessor to both read and write to these addresses. These address do not normally refer to actual RAM/ROM memory locations used by the 6510. They primarily contain the onboard registers of the I/O devices and the color RAM used by the VIC chip. The VIC (video) chip, the SID (sound) chip, the CIA's (communication) chips and the color RAM are located in this area of memory.

The VIC chip can also access (look at) memory. The VIC chip can only address 16K of memory at any one time. The VIC chip also causes the PLA to select what area of memory is available to the VIC chip. For instance, when the VIC chip wants to access the CHARACTER ROM, the PLA will select this chip rather than the I/O devices normally located from $D000-$DFFF. For our purposes, we have already covered all that we need to about the 6510 microprocessor and the PLA.

If you have a hard time digesting all the information presented to you in this chapter, DON'T WORRY ABOUT IT!!! A tremendous amount of information has been presented here. Let's just review a few of the more important concepts:

1. The 6510 microprocessor is RESET upon power up.

2. Whenever the microprocessor is RESET the LORAM, the HIRAM and the CHAREN lines will be set high.

3. The PLA will control the microprocessor's access to various areas of memory.

4. The PLA may be controlled by both hardware and software methods.

5. By grounding the EXROM line we can prevent the microprocessor from seeing RAM at locations $8000-$9FFF (very important).

6. A software RESET (SYS 64738 or JMP $FCE2) is different than a hardware RESET.

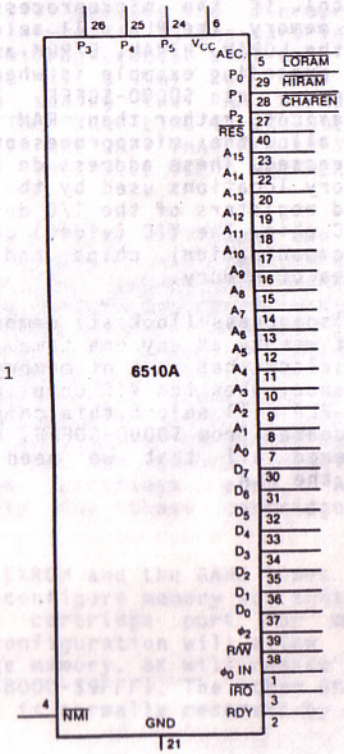


FIG 6-1

CARTRIDGES AND CARTRIDGE BOARDS

There are two main ways to use an EPROM in your Commodore computer system. You can use the EPROM on a plug-in cartridge board, or you can use it to directly replace one of the ROM chips in the computer or drive. We'll cover both of these topics, but let's concentrate on cartridges for now. There are several different kinds of cartridges for the C-64, including exotic cartridges used in some commercial products. To understand the differences among cartridges, we need to look at how cartridges are recognized and accessed by the computer. Before proceeding, be sure you have read the chapter on memory management. In that chapter we looked at the PLA and its relationship with the rest of the computer. In this chapter we'll look at how cartridges interact with the PLA.

The simplest type of cartridge is the 8K cartridge. Actually, you could put LESS than 8K of EPROM on this type of cartridge, but 8K is the maximum, so we'll loosely call it the standard 8K cartridge. A single 2764 EPROM (8K) is usually used in these cartridges (commercial cartridges may use PROMs instead). When this type of cartridge is plugged into the computer, the EPROM will be "seen" by the computer at memory address $8000-9FFF. The RAM which is normally there will "disappear". Of course, you'll get your RAM back when you unplug your cartridge. In fact, the contents of the RAM will be unchanged. (By the way, **NEVER** plug or unplug a cartridge when the computer power is on, unless you have a cartridge power switch. Doing so could destroy your computer and cartridge!)

The second type of cartridge can hold up to 16K of memory, so we'll call it the standard 16K cartridge. Two 2764 EPROMs usually supply the 16K. The first EPROM will appear in memory at $8000-9FFF, replacing the normal RAM there. The second EPROM will appear at $A000-BFFF, knocking out the BASIC ROM which is usually located there. This gives us 16K of continuous cartridge memory from $8000 to $BFFF. If you recall that the BASIC ROM is already "sitting" on top of 8K of RAM, you can appreciate how much is going on behind the scenes to keep all this straight.

The third type of cartridge is called an ULTIMAX or just MAX cartridge. ULTIMAX was a video game system produced by Commodore and sold only in Europe, and only for a short while. It used the same VIC II and SID chips as the C-64. The designers of the C-64 arranged it so that cartridges for the MAX would work on the C-64 too. On the C-64, MAX cartridges replace the KERNAL ROM located at $E000-FFFF with their own 8K of EPROM. MAX cartridges may also have another 8K of ROM memory if desired, which will appear at $8000-9FFF. One special feature of MAX cartridges is that all of the RAM of the computer disappears except for 4K at $0000-0FFF. Because we can't access most of the RAM, MAX cartridges really aren't useable in very many applications.

We just said that the EPROMs in these cartridges 'replace' different areas of memory. This is really only true for read operations. Write operations will vary in their effect depending on the type of cartridge. For example, with an 8K cartridge plugged in, the computer will be able to read the cartridge EPROM at $8000. If the computer tries to write to this location, however, the data will end up in the RAM "under" the cartridge. Likewise, with 16K cartridges write operations go to RAM automatically, even though the second EPROM at $A000-BFFF is two levels removed from RAM (the BASIC ROM is sandwiched in between the EPROM and RAM). With MAX cartridges, however, the RAM is truly "gone". Writing has NO effect on any area of RAM except the $0000-0FFF area.

How do the cartridge EPROMs and RAM chips know when to respond and when not to? Why does a read operation go to EPROM and a write operation go to RAM? We saw in a previous chapter that the C-64's PLA chip is in charge of memory management. In this chapter we'll see how the cartridge controls the PLA to produce these effects. At the same time, we'll address a related topic - what makes the three types of cartridges different? Why do the cartridge EPROMs appear at the locations they do?

The answers to these questions lie in the EPROM's **enable lines.** In order for a chip to be active, it must have a supply of power, first of all. It must also have **address and data lines** to communicate to the outside world. Most chips also have at least one **enable line.**Most of the EPROMS discussed in this book actually have two enable lines, called the **chip enable (CE)** and **output enable (OE)** lines. Both have to be controlled correctly in order to access the chip. An enable line is like a switch. The chip enable (CE) is a power switch. Even if power is available, the chip will not become active until the CE line is brought low (grounded; set to 0 volts). The abbreviation CE is usually written with a bar over it to indicate that the CE line performs its function only when it is brought low (this is called active low). When CE is held high (+5 volts), the chip is put into "standby" mode. In this mode the chip uses much less power than when active. A certain minimum amount of power is used in standby mode to keep the chip "warmed up". EPROMs don't require ANY power just to retain their data.

The other enable line, output enable (OE), controls the chip's data lines. OE is usually written with a bar over it too, since the chip will only put out data when the OE line is low. In order for the chip to be active, both OE and CE have to set low at the same time. On C-64 cartridges, the two enable lines from the chip are combined into a single line to make enabling the chip easier. The chip can be enabled by switching this one combined line high or low. Cartridges with two chips on board have a separate enable line for each chip. Each enable line is a combination of the OE and CE lines from one chip. From now on when we speak of THE enable line for a chip, we'll be referring to the combination of CE and OE.

Enable lines are used when several chips are connected to the same set of address and data lines. If more than one chip were active at the same time, there would be mass confusion (bus conflict) and possibly even physical damage to the chips. By controlling the enable lines, you can make sure only one chip at a time will be active on the address and data lines. Does this situation sound familiar? Of course - it's exactly the situation we have in the C-64 when we plug in a cartridge, since we could have EPROM, ROM and RAM potentially residing at the same address! Now we see that there is really a simple principal underlying the complexity of memory management. The PLA chip, in its infinite wisdom, knows which chip enable line to switch on, depending on what type of cartridge (if any) is plugged in and whether the operation is a read or a write. Remember, write operations are usually directed to RAM, except when writing to the I/O devices at $D000-DFFF. Read operations have to be sorted out and directed to the proper chip (EPROM, ROM or RAM).

The PLA has many lines coming into it (inputs) that it uses to sense the present state of the computer. It also has several lines coming out of it (outputs) that are used to control the memory chips. The PLA monitors its input lines continuously. Any changes in the input lines affect the output lines immediately. One input line called R/W is used to distinguish between read and write operations. Write operations are "easily" handled since they almost always go to RAM, so we'll concentrate on read operations. Of all the PLA's lines, we only need to be concerned about four right now: two input lines, GAME and EXROM; and two output lines, ROML and ROMH. The function of the GAME and EXROM lines is affected by other input lines, such as HIRAM and LORAM, but for this discussion we'll assume that HIRAM and LORAM are both held in their normal state (high).

GAME and EXROM are inputs to the PLA from the cartridge port. They are not connected to anything else in the C-64. GAME is pin #8 on the cartridge port. On a cartridge BOARD, this is the 8th pin from the left on the top side of the board (the component side, where the EPROMs are mounted). See the diagram in appendix D. EXROM is pin #9, right next to GAME. Both are active low, that is, when grounded (0 volts). With no cartridge plugged in, these lines are automatically held high (+5 volts). It's up to the cartridge to ground these lines or not, according to the memory configuration it wants the PLA to set up. In a nutshell, this is how the PLA knows what type of cartridge is connected. If EXROM alone is grounded, it indicates an 8K cartridge (regardless of how many EPROMs are actually on the board, as we'll see). If both EXROM and GAME are grounded, it indicates a 16K cartridge. Finally, if just the GAME line is grounded, it indicates a MAX cartridge (either 8K or 16K). Grounding a line is as simple as it sounds - just connect it to the computer's ground. Pins 1, 22, A and Z on the cartridge port are all grounds.

A bank-switched cartridge looks like a standard 8K cartridge to the C-64. The cartridge will ground only the EXROM line, so the PLA thinks the cartridge contains 8K of memory at $8000-9FFF. This is accurate as far as it goes: only 8K of cartridge memory will be available at a time, and it will appear at $8000. However, the cartridge board may contain any number of EPROMs. Special circuitry on-board the cartridge picks out one EPROM at a time to appear at $8000. Accessing this memory is a two-stage process: the PLA brings the ROML enable line low and then the cartridge bank-switch circuitry passes this enable signal to its currently selected EPROM. The C-64 doesn't know about the second stage, of course; it just sees a standard 8K cartridge. The only time the C-64 has to do anything special is when it wants the board to change the current EPROM.

To tell the board to change the current EPROM, we have to send a special signal to the board. We can't use the regular address, data or enable lines for this, however. Instead, most bank-switch cartridges use a special line, not normally used for anything else on the C-64 (except the Z80 CP/M cartridge). There are actually two of these lines to choose from, called **I/O1** and **I/O2**. I/O1 is set low whenever a read OR write operation accesses the $DE00-DEFF area (note that the ROML and ROMH lines are set low only on read operations). I/O2 is similiar except it's triggered by references to the $DF00-DFFF area. I/O1 and I/O2 are pins 7 and 10 on the cartridge port, respectively. On a bank-switch board, one of these lines will be connected to a special piece of circuitry called the **bank-select register (BSR)**. Depending on whether I/O1 or I/O2 is used, the BSR will appear at $DE00 or $DF00 in memory. To switch the current EPROM all you have to do is trigger the BSR, usually by writing a particular value to it. That's all there is to it. Once you trigger the BSR, the EPROM selected will appear at $8000 immediately.

Bank-switch cartridges are especially useful for programs which are too large to fit in 16K (the maximum for a regular-type cartridge). Bank-switching can also provide considerable protection for a program, depending on how it is used. There are two main ways a cartridge can use bank-switching. The simplest way is to just download the program from the EPROMs into RAM memory, one 8K chunk at a time. After downloading, many cartridges can remove themselves from memory by "ungrounding" the EXROM line via special circuitry. This frees up the RAM at $8000 (under the cartridge) for use by the program. The simple download method is relatively easy to set up but doesn't offer much protection for the program. A second way to use bank-switching is to have different sections of the program on different EPROMs. Depending on which part of the code is needed, the board can select the proper EPROM. The code is never downloaded into RAM, but rather executed directly on the EPROM. This is much more complicated for the programmer to set up, but it is also a very solid program protection technique. To make a RAM executable copy from such a cartridge, if it could be done at all, the code would have to be modified extensively.

Okay, so GAME and EXROM tell the PLA what's going on. What does the PLA do about it? This brings us to the PLA output lines ROML (ROM Low) and ROMH (ROM High). ROML and ROMH are connected only to the cartridge port. They are both normally held high (+5v) unless a cartridge grounds EXROM or GAME (or both). Depending on which of these lines are grounded, the PLA will bring ROML and/or ROMH low too. What are ROMH and ROML? Nothing more than two EPROM enable lines! ROML is the combined OE/CE enable line for the first cartridge EPROM, located at $8000-9FFF in memory. With a cartridge (any type) plugged in, the PLA will bring ROML low any time a read operation tries to access the $8000-9FFF area. Remember, bringing an enable line low will activate the chip. ROML is **not** held low **all** the time, since then the chip would be active even when other areas of memory were being accessed (resulting in bus conflict). ROML is only held low momentarily until the read operation can be performed. This is how the PLA 'tells' the EPROM it is located at $8000 - the PLA only enables the EPROM when that area of memory is being accessed.

The ROMH enable line is just a little more complicated because the second EPROM appears at different locations in memory with different types of cartridges. With a standard 8K cartridge (EXROM grounded), the second EPROM is not used and so it's disabled by holding ROMH high at all times. With a standard 16K cartridge (both GAME and EXROM grounded), any read operations in the **$A000-BFFF** area will signal the PLA to bring ROMH low. Since ROMH is connected to the second EPROM's enable lines, this will make the EPROM appear at $A000 in memory. With a MAX cartridge (GAME grounded), read operations in the **$E000-FFFF** area will enable the second EPROM through ROMH and make it appear at $E000. Note that with a MAX cartridge, you MUST have an EPROM at $E000-FFFF since the microprocessor automatically looks there on reset. The $8000-9FFF EPROM is optional with MAX cartridges (and in fact, rarely if ever used).

At this point a little review is in order. The GAME and EXROM lines run from the cartridge to the PLA. They tell the PLA what type of memory configuration to set up. Based on the memory configuration, the PLA enables the cartridge EPROMs at the proper times using ROML and ROMH. Cartridge EPROMs are only enabled for read operations, never write operations. A cartridge EPROM is only enabled when its particular area of memory is referenced. The PLA controls which area of memory is assigned to the cartridge EPROMs, depending on the state of the GAME and EXROM lines. The PLA monitors the GAME and EXROM lines continuously.

The cartridge types we've examined so far by no means exhaust the possibilities. By manipulating the GAME, EXROM and other lines, many exotic cartridges can be created. The most common example of an exotic cartridge is a **bank-switch** cartridge. "Bank-switching" means turning memory chips on and off, so different chips can occupy the same addresses at different times. Sound familiar? The C-64 already uses bank-switching, controlled by the PLA, to select its different memory configurations. What's different about bank-switched cartridges is that the bank-switching is done on-board the cartridge itself, in addition to the memory selection done by the PLA.

AUTOSTART CARTRIDGES

Suppose you wish to set up a cartridge that runs automatically when the computer is powered up or RESET. To do this you'll have to interrupt the normal power-up (RESET) process somehow, and force the computer to execute your cartridge program. Depending on where you interrupt the normal process, however, your program may have to initialize some areas of the computer for proper operation. For instance, it may have to initialize the 6510, VIC or CIA chips, or the KERNAL or BASIC RAM areas. Thus it's important to know what initialization is done normally, when it's done and why. In this chapter we'll examine the various ways to interrupt the RESET process and autostart a cartridge, including the necessary initialization tasks.

There are three main methods you can use to autostart a cartridge. Each method involves interrupting the RESET process at a different point, and each method is best suited to a different kind of cartridge. We'll start with the easiest method first, which we call the $A000 method (you'll see why in a minute). The RESET process can be divided into two phases, KERNAL initialization and BASIC initialization. BASIC initialization is only necessary if your cartridge must be compatible with the BASIC system. For example, if your cartridge adds commands to the BASIC language or uses certain BASIC ROM subroutines, you'll need to initialize BASIC.

$A0CO METHOD

If you don't need the BASIC system, you can "trick" the KERNAL RESET process into doing all your initialization for you and then autostarting your cartridge. After completing its own initialization tasks, the KERNAL RESET routine attempts to "cold-start" (initialize) BASIC. It does this by jumping to the location specified by the **BASIC COLD-START VECTOR.** Like all vectors, the BASIC cold-start vector consists of two consecutive bytes containing a memory address. The address is stored in lo-byte / hi-byte order, which means the low order (least significant) byte is first and the high order (most significant) byte is second. The KERNAL expects the BASIC cold-start vector to be found in locations $A000-01, which is normally at the very beginning of the BASIC ROM. The contents of these two locations in the BASIC ROM are $94 and $E3 respectively, which means they "point" to location $E394 (vectors are also called pointers). This location is the start of the BASIC cold-start routine.

If we could change the BASIC cold-start vector, we could make it point to our cartridge program. Our cartridge would start up automatically after all KERNAL initialization was finished. But since this vector is in the BASIC ROM, how do we change it? Answer: replace the BASIC ROM. Not physically, of course, but by using a standard 16K cartridge. Recall that standard 16K cartridges reside at $8000-BFFF. The PLA switches out the BASIC

ROM and selects the 16K cartridge configuration when it senses that both of the GAME and EXROM lines are grounded. Two 8K EPROMS are required for 16K of memory on the cartridge. The first EPROM resides at $8000-9FFF and the second EPROM resides at $A000-BFFF. All you have to do is put a vector at $A000-01 which points to the beginning of your program, and the cartridge will be started automatically at the end of KERNAL initialization.

If you only need 8K for your cartridge, you can still use this technique. Just use the second EPROM ($A000-BFFF) and leave the first EPROM ($8000-9FFF) socket empty. As long as GAME and EXROM are grounded, the PLA will still choose the 16K configuration. This means the RAM normally at $8000-9FFF will still be switched out, however (and the BASIC ROM too, of course). Since you're not using the first cartridge EPROM, you'll have a "hole" in memory from $8000-9FFF. If you try to read from this area, random data may appear there. You may even be able to use this phenomena as part of a protection scheme. As usual, data written to this area will be placed in the underlying RAM, even though you can't read it back out.

So, to review a bit, the $A000 method is the easiest autostart technique to use because all KERNAL initialization is done for you. You don't have to worry about BASIC initialization since you can't use BASIC anyway (the BASIC ROM is switched out). Of course, you don't have access to the BASIC ROM subroutines either (there's a lot of useful stuff in there). This makes the $A000 method most suitable for cases where you need the maximum 16K of cartridge memory, or where you only need 8K and don't need BASIC.

CBM80 METHOD

The second cartridge autostart method is by far the most common. It can be used by both 8K and 16K cartridges that reside at $8000. Although this method is sometimes called the "cartridge autostart option", it can be used equally well by RAM-based programs, and often is. You probably know it as the "CBM80" method. One of the first things the KERNAL RESET routine does is check locations $8004-08 for the string of characters "CBM80". If these exact characters are **NOT** found there, the KERNAL RESET process continues normally.

If the "CBM80" **IS** found, the RESET routine is interrupted and the processor immediately jumps to whatever location is specified by the **CARTRIDGE COLD-START VECTOR.** This vector is expected to be found at locations $8000-01. You must place a pointer here, in standard lo-byte / hi-byte order, directing the processor to the beginning of your cartridge code. From that point on, your cartridge must handle all the initialization itself for any functions it will use, such as the I/O devices or KERNAL or BASIC routines. Fortunately, you still have the KERNAL initialization routines available for use. Unless you know exactly what you are doing, your cartridge should use these routines to initialize the functions it needs.

Figure 8-1 presents a "generic" cartridge initialization routine. This routine duplicates most of the normal RESET process. In fact, it's taken right from the main parts of the KERNAL ($FCEF-FE) and BASIC ($E394-9F) RESET routines. This generic routine will be adequate for 99% of all cartridges.

Figure 8-1: CBM80 Cartridge Initialization

```
8000  09 80                 Cartridge cold-start vector = $8009
8002  25 80                   "   "   "  warm   "   "   "  =  8025
8004  C3 C2 CD 38 30        CBM80 - Autostart key

                            KERNAL RESET Routine
8009  8E 16 D0    STX $D016 Turn on VIC for PAL/NTSC check
800C  20 A3 FD    JSR $FDA3 IOINIT - Init CIA chips
800F  20 50 FD    JSR $FD50 RAMTAS - Clear/test system RAM
8012  20 15 FD    JSR $FD15 RESTOR - Init KERNAL RAM vectors
8015  20 5B FF    JSR $FF5B CINT   - Init VIC and screen editor
8018  58          CLI       Re-enable IRQ interrupts

                            BASIC RESET  Routine
8019  20 53 E4    JSR $E453 Init BASIC RAM vectors
801C  20 BF E3    JSR $E3BF Main BASIC RAM init routine
801F  20 22 E4    JSR $E422 Power-up message / NEW command
8022  A2 FB        LDX #$FB
8024  9A          TXS       Reduce stack pointer for BASIC

8025  ........    ........  START YOUR PROGRAM HERE
```

The cartridge cold-start vector and autostart key (CBM80) have already been discussed. The warm-start vector at $8002-03 is a feature that allows you to re-enter your program after a full initialization has already been done. Once a cold-start has been done, it usually doesn't need to be done again. Pressing the RESTORE key calls the NMI routine (NON-MASKABLE INTERRUPT), which will see the CBM80 and jump to the location indicated bt the warm-start vector. This is why many programs restart themselves when you press the RESTORE key. In our initialization routine we have pointed the warm-start vector to the start of your program; you could also point it to $8009 to perform a full cold-start on RESTORE. If you want to disable the RESTORE key entirely, point the warm-start vector to $FEBC (return from NMI).

We have included the BASIC RESET process in this cartridge initialization routine too. Actually, the normal BASIC RESET routine dead-ends with a jump to the BASIC direct mode interpreter, also known as "READY" mode. This prints the "READY." prompt and then sits there waiting for you to type a BASIC command. You won't usually want to exit into READY mode at this point since BASIC will take over and your cartridge will lose control. If you do want to exit to BASIC now or later, you may do so with JMP $E386. By the way, the routine called at $801F (JSR $E422) prints the normal power-up screen and does a NEW command. If you want to skip the power-up message, just call the NEW command directly using JSR $A644 instead of JSR $E422.

To summarize, the CBM80 method can be used with either 8K or 16K standard cartridges (which start at $8000). The cartridge initialization routine above will be sufficient for the vast majority of cartridges. KERNAL initialization must be done at least once (on power-up or RESET). BASIC initialization can be skipped if you're not using BASIC, and **MUST** be skipped if you're using a 16K cartridge. Through the cartridge warm-start vector, the RESTORE key can be set up to re-enter your program or it can be disabled entirely. The CBM80 method is by far the most common cartridge autostart method.

MAX METHOD

The third and last autostart method is the MAX method. This requires the use of an ULTIMAX cartridge (one that grounds just the GAME line). The second EPROM in a MAX cartridge resides at $E000-FFFF, replacing the KERNAL ROM (the first EPROM appears at $8000-9FFF if used). MAX cartridges have many limitations and are rarely used commercially except for simple video games. One limitation of MAX cartridges is that only 4K of the computer's RAM is available for use, and one-quarter of that is required for screen memory. Another limitation is that the KERNAL ROM is switched off, which means you must write your own powerup/RESET initialization routines (although you may use the KERNAL routines as a guide). You don't need to worry about BASIC initialization since the BASIC ROM is switched out too. For these reasons, you'll probably never need to set up a MAX cartridge. Only an advanced user would want to consider using this method.

MAX cartridges autostart through a hardware function of the 6510 processor rather than through a software routine in the KERNAL like the other two methods. When the C-64 is powered up, a special circuit RESETs the 6510 microprocessor, SID and CIA chips, just as if you had performed a hard RESET yourself (with a RESET button). Two very important events take place immediately, before any instructions are even executed. First, the I/O devices (VIC, SID, CIAs and color RAM) are switched into memory by the PLA, as well as the KERNAL and BASIC ROMs (normally). Second, the 6510 processor fetches its **RESET vector** from locations **$FFFC-FD**, normally in the KERNAL ROM. The RESET vector is a two-byte value that points to the beginning of the KERNAL RESET routine. The processor **ALWAYS** gets its RESET vector from locations $FFFC-FD. This feature is "hard-wired" into the 6510 chip itself, and **CANNOT** be changed! You must make sure the processor finds a valid address in locations $FFFC-FD, which is why the KERNAL is normally switched in first on RESET. The processor does an indirect jump based on the RESET vector and normally begins executing the KERNAL RESET routine.

The only alternative to executing the KERNAL RESET routine is if a MAX cartridge is present. If the PLA senses that a MAX cartridge is plugged in (GAME line grounded), it switches in the cartridge at $E000-FFFF, replacing the KERNAL. Then when the 6510 fetches its RESET vector from $FFFC-FD, it will get it from the cartridge instead of the KERNAL. You simply place a vector at $FFFC-FD pointing to the beginning of your RESET routine, and the processor will start executing your routine automatically. This is how the MAX autostart method works - your cartridge is switched in and grabs control right from the start.

Now it's up to your cartridge to perform all necessary initialization. For instance, the VIC chip must be initialized in order to use the screen. Likewise, other I/O devices will have to be initialized if you want to use the keyboard, the joysticks, the sound chip, IRQ interrupts, etc. Initializing and controlling these devices can be quite complicated. We recommend that you use the corresponding KERNAL routines as a model for your own routines. In fact, many MAX cartridges contain almost byte-for-byte copies of KERNAL routines. While we can't cover the KERNAL routines in depth in this book, we can summarize the various initialization routines used in the normal RESET process. This will help get you started towards an understanding of what initialization your cartridge will require.

NORMAL RESET PROCESS

Recall that when the 6510 processor is RESET, it begins executing at the address specified by the RESET vector at $FFFC-FD. In the KERNAL ROM, these locations contain a pointer to the KERNAL RESET routine at $FCE2 (decimal 64738). The main part of the KERNAL RESET routine is shown in figure 8-2. Note the similarities to our CBM80 cartridge initialization routine (fig. 8-1). The KERNAL RESET routine takes three important steps right away. First, IRQ interrupts are disabled with a SEI instruction, so the routine won't be interrupted. Second, the stack pointer is initialized by transferring a value to it from the X-register, using a TXS instruction. The stack pointer indicates the next empty position on the stack, which grows DOWNWARD in memory from $01FF to $0100 (i.e. the pointer decreases as the stack is filled). The stack pointer contains a random value on RESET, so we should put a value here before any stack operations take place (i.e. a PHA, PLA, PHP, PLP, JSR, RTS, RTI instruction or an NMI, IRQ or BRK interrupt). The RESET routine sets the stack pointer to $FF, which starts the stack out at the very top (allowing it the maximum space).

The third step is to clear the decimal mode flag with a CLD instruction. This flag controls whether math instructions like ADC (add) and SBC (subtract) are performed in normal "hex" format (actually binary) or in BCD format (BINARY CODED DECIMAL). Like the stack pointer, this flag has a random value on power-up, so it is set to 0 to ensure that math is done in hex format. If you write your own initialization routine for a MAX cartridge you should also do these three things right away.

Figure 8-2: KERNAL RESET ROUTINE

```
FCE2  A2 FF     LDX #$FF      Stack pointer value
FCE4  78        SEI           Disable IRQ interrupts
FCE5  9A        TXS           Initialize stack pointer
FCE6  D8        CLD           Clear decimal mode
FCE7  20 02 FD  JSR $FD02     Check for CBM80 key
FCEA  D0 03     BNE $FCEF     Branch if not found
FCEC  6C 00 80  JMP ($8000)   Jump to cartridge cold-start
FCEF  8E 16 D0  STA $D016     Turn on VIC (Aç=$05)
FCF2  20 A3 FD  JSR $FDA3     IOINIT - Init CIA chips
FCF5  20 50 FD  JSR $FD50     RAMTAS - Clear/test system RAM
FCF8  20 15 FD  JSR $FD15     RESTOR - Init KERNAL RAM vectors
FCFB  20 5B FF  JSR $FF5B     CINT - Init VIC and screen editor
FCFE  58        CLI           Re-enable IRQ interrupts
FCFF  6c 00 A0  JMP ($A000)   Jump to BASIC cold-start ($E394)
```

After the first three steps, the RESET routine calls an important subroutine at $FD02. This routine checks locations $8004-08 for "CBM80" autostart key. If these exact characters are found there, the cartridge cold-start vector is fetched from $8000-01. Execution continues at whatever location is indicated by this vector. This is the point at which the CBM80 autostart method takes control.

If there is no "CBM80" found, the RESET process continues at $FCEF. The X register is stored into location $D016, which is the VIC control register. The value of X at this point is always $05 or less, since it was used as an index in the check for "CBM80" (which has 5 characters). Commodore says it's extremely important to make sure bit number 5 of this value is a 0, which it is in this case. A 0 in bit 5 supposedly turns the VIC chip on and a 1 turns it off (see p. 322 and p. 448 in the Prog. Ref. Guide). For safety's sake your own initialization routine should set this bit to 0 too. There's an interesting side-effect when values less than $05 are stored in this register. In those cases, bit 3 will also be a 0, which selects 38-column mode. That's why the screen "shrinks" when the computer goes through its normal RESET process - bet you always wondered about that!

Next, the four main KERNAL initialization routines are called. These are the same routines we called in our CBM80 initialization routine. The first routine is IOINIT, located at $FDA3. This routine can also be reached by jumping to $FF84 in the standard KERNAL jump table. IOINIT initializes the CIA chips. It also does some other minor initialization such as turning off the SID's sound, switching in the BASIC and KERNAL ROMs (redundant on a hard RESET) and sending a high clock signal ("1" bit) on the serial bus. Next, the KERNAL routine RAMTAS at $FD50 is called ($FF87 in the jump table). This routine clears and tests RAM. First, the routine fills locations $0002-0101, $0200-02FF and $0300-$03FF (pages 0, 2 & 3) with $00 bytes. This piece of code is responsible for the cassette buffer, etc. being cleared on RESET. Note that the stack is not cleared (except the bottom two bytes).

After this, RAMTAS sets the cassette buffer pointer and then begins testing RAM memory starting at $0400 (the screen). The purpose of this test is to find the start of non-RAM memory, i.e., to see if there is cartridge ROM at $8000. The test is supposed to be "non-destructive" in that RAM memory is not altered. First, the current contents of the location to be tested are saved in the X-register. A $55 byte is stored into the location and then the location is read back to see if the $55 was stored successfully. If the location now contains a $55 then it is obviously in RAM - or is it? What if the location is in ROM but happened to already contain a $55? To doublecheck this, the process is repeated with the value $AA instead of $55. If it passes both tests, the location is definitely in RAM. The original value saved in X is restored, and the test continues with the next byte.

Eventually, the routine will run into ROM (either cartridge ROM at $8000 or BASIC ROM at $A000). When ROM is encountered, a very undesirable side-effect occurs. The $55 byte that is written out goes into the RAM "under" the ROM, wiping out the value that was there. This is important to remember when you are trying to recover a crashed program by resetting it. Once the start of ROM is found, a routine at $FE25 (MEMTOP, jump table $FF99) is called to set the top of system RAM to the beginning of ROM. The top of system RAM is used in the calculation of the number of BASIC bytes free. Finally, the bottom of system RAM is set to $0800, and the start of the screen is set to $0400 for the screen editor.

Back in the main RESET routine, a routine at $FD15 is called (RESTOR, jump table $FF8A). This routine copies the KERNAL's indirect RAM vectors to $0314-33 from the table at $FD30-4F. Oddly enough, it also copies this vector table into the RAM under the KERNAL at $FD30-4F too! If you are trying to recover the contents of the RAM under the KERNAL after a RESET, you should remember this feature.

The final KERNAL initialization routine is at $FF5B (CINT, jump table $FF81). This routine initializes the VIC chip and screen editor variables. The VIC chip is initialized by calling a routine at $E5A0 which downloads a set of constants to the VIC from $ECB9-E6. This sets the border and background colors as well as the raster interrupt register, used in the PAL/NTSC check discussed below. The screen editor is initialized by a routine at $E518. This sets the character color, keyboard decode table vector, cursor blink and key repeat rates, and then clears the screen. The CINT routine ends with the PAL/NTSC check. NTSC is the North American TV standard and PAL is the International TV standard. There are more lines on the screen with PAL. This fact is used to detect which system you have, so the IRQ and RS-232 timing can be adjusted accordingly. The VIC raster (screen line) interrupt was set earlier to occur on a line which doesn't exist with NTSC. Later (at $FF63) the interrupt is checked to see if it happened. If it did, we're on a PAL system, otherwise it's NTSC. See the Prog. Ref. Guide pp. 150 & 447 for more information on the raster interrupt register.

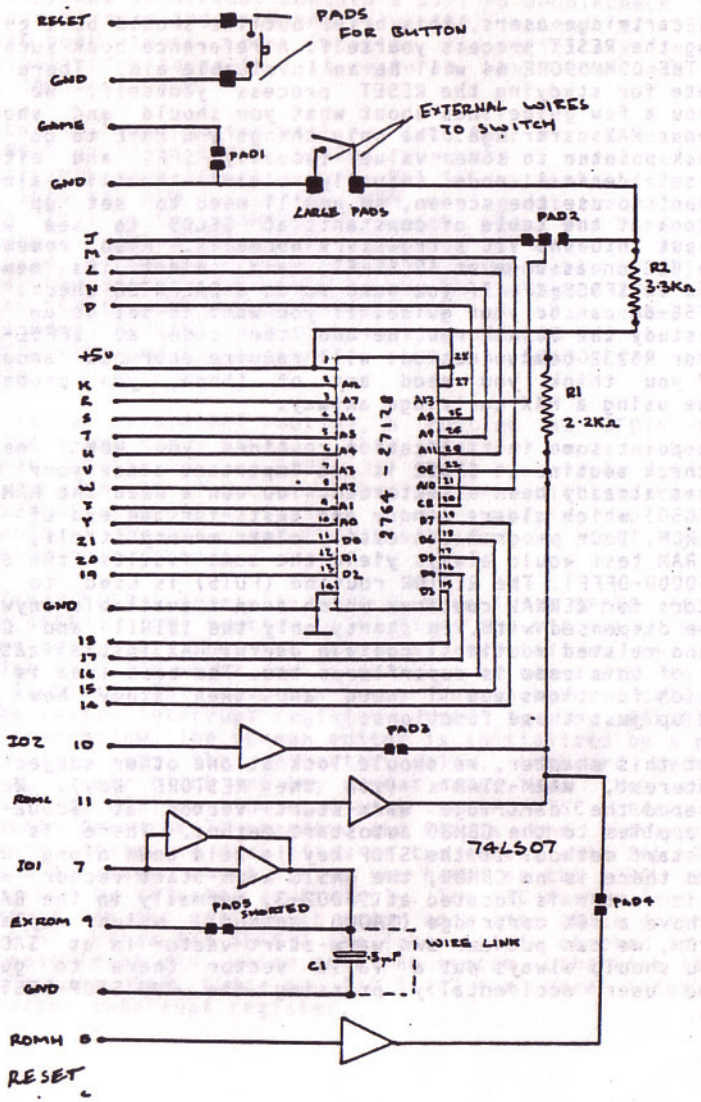
That's about it for the KERNAL RESET routine. IRQs were disabled earlier, so they are re-enabled with a CLI instruction. Finally, BASIC initialization is begun by jumping based on the BASIC cold-start vector in the BASIC ROM at $A000-01. If a 16K standard cartridge is present, however, its second EPROM will have replaced the BASIC ROM in memory. In this case we must put a vector at $A000 (in the second cartridge EPROM) to point to the start of the cartridge program. This is the basis for the $A000 autostart method. Notice how all the KERNAL initialization has already been done for us in this case.

For you MAX cartridge users, this brief outline should be a guide to examining the RESET process yourself. A reference book such as ANATOMY OF THE COMMODORE 64 will be an invaluable aid. There is no substitute for studying the RESET process yourself. We can only give you a few guidelines about what you should and should not do in your MAX cartridge. The only things you HAVE to do are set the stack pointer to some value (usually $FF) and either clear or set decimal mode (usually clear). You'll almost certainly want to use the screen, so you'll need to set up the VIC chip. Look at the table of constants at $ECB9 to see what values are put into the VIC's registers normally. Also remember to turn the VIC on as done at $FCEF-F1, and select its memory bank as done at $FDCB-CF. If you want to do a PAL/NTSC check, the code at $FF5E-6A can be your guide. If you want to set up an IRQ interrupt, study the IOINIT routine and the code at $FF6E-7C. Disk, tape or RS232 communications will require enormous amounts of code. If you think you need any of these, you probably shouldn't be using a MAX cartridge anyway.

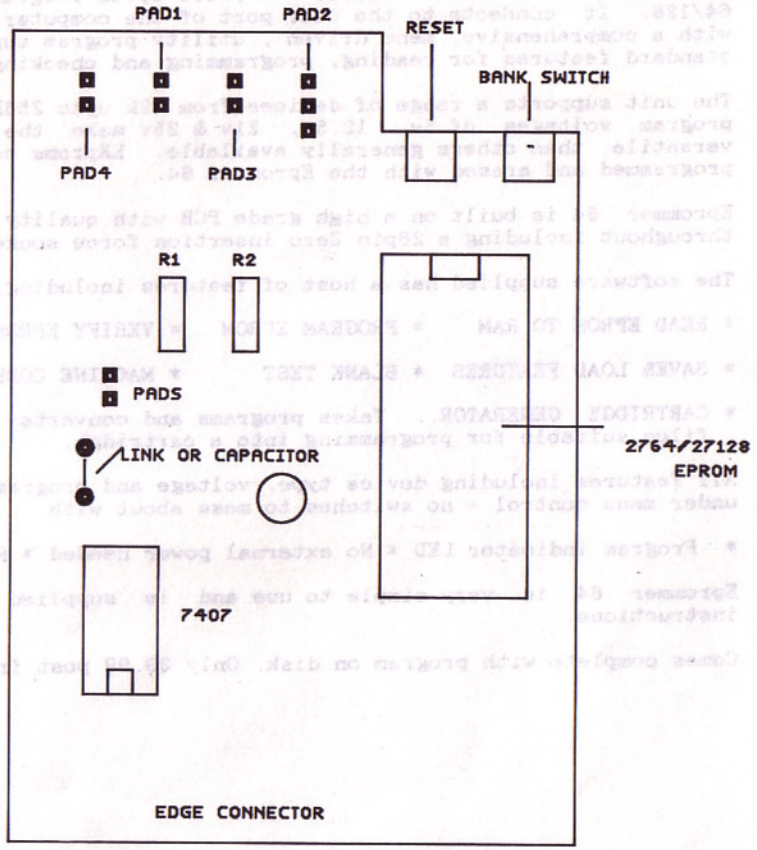
We can also point some initialization routines you WON'T need. The CBM80 check routine at $FD02 is not important since your MAX cartridge has already been autostarted. You won't need the RAMTAS routine ($FD50) which clears memory and tests for the end of RAM / start of ROM. Your program can easily clear memory itself, and the end of RAM test would always yield the same results (the only RAM is at $0000-0FFF). The RESTOR routine (FD15) is used to set up RAM vectors for KERNAL routines which aren't available anyway, so it can be dispensed with. In short, only the IOINIT and CINT routines (and related routines) contain useful MAX initialization code. A lot of this code is superfluous too. The best idea is to plan out which functions you'll need and then study how the KERNAL sets up just those functions.

To round out this chapter, we should look at one other subject of general interest, WARM-STARTs (via the RESTORE key). We've already covered the cartridge warm-start vector at $8002-03, which only applies to the CBM80 autostart method. There is one other warm-start method. If the STOP key is held down along with RESTORE, and there is no CBM80, the BASIC warm-start vector will be used. This vector is located at $A002-3, normally in the BASIC ROM. If we have a 16K cartridge ($A000 method), which replaces the BASIC ROM, we can put our own warm-start vector in at $A002. In fact, you should always put a valid vector there to guard against the user accidentally pressing the RUN/STOP-RESTORE combination.

Finally, there is no warm-start method for MAX cartridges unless you program it yourself. You may choose to use the RESTORE key for this or some other method. If you don't use the RESTORE key, you should set the 6510 NMI vector at $FFFA-FB to point to an RTI instruction in case the user accidentally presses RESTORE. The 6510 is hard-wired to use $FFFA-FB as its vector when an NMI is generated (RESTORE key pressed), just as it always uses $FFFC-FD for its RESET vector.



CARTRIDGE BOARD LAYOUT



EPROMMER 64

Eprommer 64 is a multi feature, low price eprom programmer for the 64/128. It connects to the user port of the computer and comes with a comprehensive, menu driven , utility program supporting all standard features for reading, programming and checking eproms.

The unit supports a range of devices from 32k upto 256k. Multiple program voltages of 5v, 12.5v, 21v & 25v make the unit more versatile than others generally available. EEproms can also be programmed and erased with the Eprommer 64.

Eprommer 64 is built on a high grade PCB with quality components throughout including a 28pin Zero insertion force socket.

The software supplied has a host of features including;

* READ EPROM TO RAM * PROGRAM EPROM * VERIFY EPROM

* SAVE& LOAD FEATURES * BLANK TEST * MACHINE CODE MONITOR

* CARTRIDGE GENERATOR.. Takes programs and converts them into files suitable for programming into a cartridge.

All features including device type, voltage and program selections are under menu control - no switches to mess about with.

* Program indicator LED * No external power needed * No more to buy

Eprommer 64 is very simple to use and is supplied with full instructions.

Comes complete with program on disk. Only 39.99 post free.

DATEL ELECTRONICS

DATEL ELECTRONICS

DATEL
ELECTRONICS

DATEL
UTILITIES

GOVAN ROAD
FENTON INDUSTRIAL ESTATE
STOKE-ON-TRENT
STAFFS
Tel: 0782 744707
Fax: 0782 744292