EXPLORER

MG

Manual written by Craig G. Miller

Explorer Software and Associated Manual
(Printed in the United States of America)

TABLE OF CONTENTS

The EXPLORER - An Overview

The Explorer program was designed to be used as a tool to help you understand how your computer thinks and operates and to be as transparent as possible to the environment or program that loaded it. The Explorer converts your 4A into a programmer's instrument similar to a engineers Logic State Analyzer. But, instead of high/low trace lines on a screen it will display a number of items pertinent to the execution of programs by the 9900 Microprocessor.

This new Explorer Instrument will allow you to execute Extended Basic, Basic, Assembly Language and a variety of Command Modules all under your control. You can stop and start execution at any time with just the press of a key. You can watch the actual program screen in slower motion or you can watch the Explorer's Main Screen as it is dynamically updated after each and every instruction. You can stop the program and examine and modify memory and other items, which allows you to conduct "WHAT IF" experiments.

The heart of the Explorer is a Machine Language Interpreter that thinks, acts and has the same logic as a software based 9900 microprocessor. Through this interpreter the Explorer will open the window into your 99/4A and allow you to see the actual inner workings of your computer in action.

The Explorer goes through great lengths to analyze and preserve the environment that it is loaded into so that you can easily continue execution of that program or module. It is fully compatible with the Extended Basic, Editor Assembler, Mini Memory and other Assembly Language Loaders to allow a wide variety of Explorations.

The following few pages contain the instructions and examples for loading the Explorer into these environments. We hope that you find this new Explorer instrument to be as much of an education in the the 99/4A's operation system, GPL Interpreter, and other languages and modules as we have.

THE EXPLORER FILES

MGEX00Z00 - This is the serial number of your disk for the warranty registration. It will appear as the first item on a Catalog list.

EXP - This is the Editor/Assembler, Mini Memory, Basic through the E/A or Mini Mem, or other Load and Run type loaders version of the Explorer.

MEXP - This is the Myarc Disk Controller CARD* version of the above file.

XBEXP - This is the Extended Basic loader version of the Explorer.

MXBEXP - This is the Myarc Disk Controller CARD* version of the above file.

BASAMPLE - This is a sample Basic program that loads the Explorer and allows you to continue execution of the Basic program.

XBSAMPLE - This is a sample Extended Basic program that loads the Explorer and allows you to continue execution of the Extended Basic program.

XBLOAD - This file can be renamed as LOAD so that whenever the Explorer disk is in drive 1 and Extended Basic is selected the Explorer will automatically load and pass control to you.

XBMEMFULL - This is an Extended Basic MERGE type file that contains line number 1. This file allows you to bypass the Extended Basic loader's Memory Full error condition when you or your Extended Basic program has previously loaded another Assembly language program that has filled up LOW Memory expansion.

XBMEMOFF - This file is an example of how you can turn off memory expansion to load and execute your Extended Basic program from VDP RAM instead of High Memory Expansion. This is useful when your Extended Basic program is larger than the approx 6K bytes left after the Explorer is loaded.

NOTE

If you have a Myarc Disk Controller CARD replace the reference to "DSK1.EXP" in the BASAMPLE file with "DSK1.MEXP". Also replace the references to "DSK1.XBEXP" in the XBSAMPLE, XBLOAD and XBMEMFULL files with "DSK1.MXBEXP".

* The Explorer is compatible with the Myarc Disk Controller CARD only. It will not work with the Myarc MPES (Mini Peripheral Expansion System) due to low level hardware differences between the Card and this system.

LOADING THE EXPLORER INTO THE

EXTENDED BASIC ENVIRONMENT

The Explorer can be loaded into the Extended Basic environment either from the Command/Edit mode or from a running Extended Basic Program. If it is loaded from the Command/Edit mode of Extended Basic then, when you start up the Explorer, it will return to the Command/Edit mode environment. If it is loaded from a running program then, when you start up the Explorer, it will continue execution of the program exactly where it left off.

NOTE

CALL INIT must be executed either by you or the running Extended Basic program sometime prior to executing the CALL LOAD("DSK1.XBEXP"). Also, if you have a Myarc Disk Controller Card replace all references to "DSK1.XBEXP" with "DSK1.MXBEXP".

FROM COMMAND/EDIT MODE

Type in CALL LOAD("DSK1.XBEXP") and press enter. The Explorer will load and pass control of your computer to you. When you start up the Explorer it will return to the Command/Edit mode of Extended Basic but now it will be in your control. Once the Application Program screen has scrolled up 1 line and the cursor has reappeared you can slowly type in anything that is valid for the Extended Basic Command/Edit mode and watch it work.

FROM A RUNNING PROGRAM

Place CALL LOAD("DSK1.XBEXP") into your Extended Basic program, where you want the Explorer to load and pass control to you, and then RUN your Extended Basic program. When your Extended Basic program reaches the line that contains this CALL LOAD the Explorer will load and pass control of your computer to you. When you start up the Explorer it will continue executing your Extended Basic program where it left off but it will be in your control so you can watch it work.

Type in OLD DSK1.XBSAMPLE and LIST and RUN it to see an example of this type of loading and program execution. After this program has finished executing, the Extended Basic interpreter will go through its READY routine. This will load some default values into the Vdp registers, restore the color table and character set and the scroll the screen up one line to place the * READY * message on the Application Program screen. Then it will scroll the screen up 2 more lines and bring out the cursor. At this point Extended Basic is back in Command/Edit Mode and is waiting for you to type something in.

EXTENDED BASIC PROGRAM SIZE

The Explorer occupies approximately 18K bytes of High Memory Expansion when it is loaded. This leaves all of Low Memory Expansion free for your Assembly Language subprograms and part of High Memory Expansion free for your RUNNING Extended Basic program. If you are not sure if your Extended Basic program will fit in memory with the Explorer then load and check your program as follows:

1. OLD DSKx.yourprogram
2. Type in RUN - press Enter and then Press and hold down FCTN 4 CLEAR.
3. When the program breaks - type in SIZE and press Enter.
4. The STACK size doesn't matter since this is in Vdp Ram.
5. The PROGRAM space must be at LEAST 18,400 Bytes Free for the Explorer to execute your Extended Basic program properly.

The reason you Must RUN and break your program before checking its size is to allow Extended Basic to perform the Pre-Scan routine. During Pre-Scan Extended Basic reserves room for your variables. The string variables stay in Vdp Ram so they don't matter. However, the Numeric variable's values are stored in High Memory Expansion so you must have enough space for these or your Extended Basic program will not execute properly. NOTE: We don't really recommend that you run large Extended Basic programs through the Explorer because of the amount of time that it takes. The Explorer was meant to be used with direct CALL's and small programs.

If your Extended Basic program is too large to work in Memory Expansion with the Explorer you can use the XBMEMOFF file to turn off Memory Expansion and load your program into Vdp Ram. By doing this you can execute an Extended Basic program that has a RUNNING size of up to 12,876 bytes with CALL FILES(1). This also has an added advantage in that you can easily follow the Extended Basic interpreter's accesses to your program by having the Explorer's Memory Window set on Vdp Ram and in Dynamic mode. (see Memory Windows for more information). To use this file simply place your Program name and drive location in line 2 of this file instead of the DSK1.XBSAMPLE that is currently there. Also, your program MUST contain the CALL LOAD("DSK1.XBEXP") statement so that IT loads the Explorer.

A MEMORY FULL ERROR CONDITION

If, when you try to load the Explorer, you receive a MEMORY FULL error message on the screen you may be able to use the XBMEMFULL file to bypass this condition. This file is a MERGE type file that contains a single line (line number 1). This file will only work properly IF the Assembly Language program that was previously loaded, and is causing this error condition, resides ENTIRELY in LOW MEMORY EXPANSION. If any part of it resides in High Memory Expansion the Explorer may overwrite it when it is loaded.

This error condition arises when the difference between the First Free address and Last Free address in Low Memory Expansion is too small. So, this file saves the current First Free and Last Free addresses in Low Memory Expansion and then loads the the default values for these pointers, loads the Explorer into High Memory and then restores the save values for these pointers when the Explorer is started up.

Once again, this will NOT work if the Assembly Language program that was previously loaded is NOT entirely in Low Memory Expansion or if any portion of it resides in the Explorer's program space.

Here is a break down of this one line file:

CALL PEEK(8194,_0,_1,_2,_3)::	Save the current First Free Address in _0 & _1 Save the current Last Free Address in _2 & _3
CALL LOAD(8194,36,250,64,0,	Put default value of >24FA into FFA Put default value of >4000 into LFA
"DSK1.XBEXP",	Load the Explorer
8194,_0,_1,_2,_3)	Restores the saved values into FFA & LFA When you start up the Explorer.

NOTE: The Extended Basic Versions (XBEXP & MXBEXP)) will only work in Extended Basic. You cannot use this version with the Editor/Assembler or Mini Memory modules.

LOADING THE EXPLORER INTO THE

BASIC ENVIRONMENT

With either the Editor Assembler or the Mini Memory module or with the Myarc Disk Controllers CALL for loading Assembly Language programs the Explorer can be loaded into the Basic environment either from Command/Edit mode or from a running Basic Program.

NOTE

CALL INIT does NOT have to be executed prior to loading the Explorer with these modules unless you receive a MEMORY FULL error condition. The CALL LOAD statement in these modules automatically executes the CALL INIT if it hasn't been previously executed. Also any references to DSK1.EXP should be replaced with DSK1.MEXP to load the Explorer with the Myarc Disk Controller Card.

FROM COMMAND/EDIT MODE

Type in CALL LOAD("DSK1.EXP") and press enter. The Explorer will load and pass control of your computer to you. When you start up the Explorer it will return to the Command/Edit mode of Basic but now it will be in your control. Once the Application Program screen has scrolled up 1 line and the cursor has reappeared you can slowly type in anything that is valid for the Basic Command/Edit mode and watch it work.

FROM A RUNNING PROGRAM

Place CALL LOAD("DSK1.EXP") into your Basic program, where you want the Explorer to load and pass control to you, and then RUN your program. When your Basic program reaches the line that contains this CALL LOAD the Explorer will load and pass control of your computer to you. When you start up the Explorer it will continue executing your Basic program where it left off but it will be in your control so you can watch it work.

Type in OLD DSK1.BASAMPLE and LIST and RUN it to see an example of this type of loading and program execution. After this program has finished executing, the Basic interpreter will go through its DONE routine. This will scroll the screen up one line and place ** DONE ** on the screen, then it will scroll the screen up one more line and then load some default values into the Vdp registers, restore the color table and character set and finally scroll the screen up one more line and bring out the cursor. At this point Basic is back in Command/Edit Mode and is waiting for you to type something in.

BASIC PROGRAM SIZE

Unlike Extended Basic, the size of your Basic program is not critical because Basic programs are ALWAYS loaded and executed from Vdp Ram. They are never run out of Expansion Memory so ALL of Expansion Memory is free for Assembly Language programs.

MEMORY FULL ERROR CONDITION

This condition may occur if you have previously loaded an Assembly Language program into memory. If this condition occurs you can easily clear it by executing CALL INIT. Unfortunately this will also clear out all references to the previously loaded Assembly Language program so you can not execute it through the Explorer.

The Editor/Assembler - Mini Mem version of the Explorer (DSK1.EXP & DSK1.MEXP) loads into High Memory Expansion and occupies approximately 18K bytes of this memory area. It does not use Low Memory Expansion or the Mini Mem Ram so these areas and approx 6K bytes of High Memory are left free for your Assembly Language Subprograms.

LOADING THE EXPLORER INTO THE

EDITOR ASSEMBLER or MINI MEMORY ENVIRONMENTS

The Explorer can be loaded alone or along with a Non-Auto-Start Assembly Language program or subprogram. The Explorer MUST be the LAST file loaded because it will take control. Loading your own Assembly programs along with the Explorer will allow you to use the Explorer as a very powerful debugging aid.

Once again, the Explorer loads itself into High Memory Expansion and occupies 18K bytes of this area. This leaves 6K bytes free for your assembly program in High Memory plus room for the XOP 1 instruction. This also leaves all 4K of the Mini Memory Ram for your use and ALL 8K of Low Memory Expansion. The Explorer does NOT use any of the Editor Assembler Utilities so your program can write over these or modify them to suit your needs. With a little fancy AORGing in your own Assembly Language program you can easily load up to 18K bytes and still have enough room for the Explorer.

EDITOR ASSEMBLER ENVIRONMENT

Select 3 LOAD AND RUN

```
When this prompt appears   * LOAD AND RUN *
                           FILE NAME
Type in                    DSK1.EXP         and press Enter
```

The Explorer will load and pass control of your computer to you.

MINI MEMORY ENVIRONMENT

Select 1 LOAD AND RUN

```
When this prompt appears   * LOAD AND RUN *
                           FILE NAME
Type in                    DSK1.EXP         and press Enter
```

The Explorer will load and pass control of your computer to you.

LOADING THE EXPLORER INTO

OTHER LOAD AND RUN TYPE LOADER ENVIRONMENTS

The Explorer can also be loaded with either the Myarc Disk Controller Card's CALL LR("DSK1.MEXP") for Load and Run Type files or from our Disk Manager for the Corcomp Disk Controller. When the Myarc Disk Controller's CALL LR is executed from Basic it does not require the Editor/Assembler or Mini Mem modules to be plugged in. This leaves your module port free for Explorations of Command Modules (see Explorations for more info). Also, when you select Load and Run Assembly file from within our Disk Manager program you can have any module you would like to Explore plugged into the cartridge port. With this ability it opens up a wide variety of Explorations such as, Parsec, Adventure, Personal Record Keeping, Munch Man, Number Magic and on and on and on. There goes the sleep!

The Myarc CALL LR is executed from Basic so when you start up the Explorer it will return to Basic. If you are trying to access a command module then set the Explorer to execute the Power Up routine (WS = 83E0 PC = 0024). This will allow you to access the module through the normal menu screen when you get there. Our Disk Manager loader is executed from within the Disk Manager program and as such the Disk Manager is written over when it loads another assembly language program. This means that there isn't any place to return to. So, before you start up the Explorer, after loading it with this loader, change the Cpu WS and PC to the values for the Power Up routine (see Explorations for more info), this gives the Explorer someplace to go to and allows you to select a module from the menu.

MYARC LOADER FROM BASIC

Type in CALL LR("DSK1.MEXP") and press Enter.

DISK MANAGER LOADER

```
Select          1 File Utilities

then select     2 Load and Run Assembly file

Input           Drive No. : 1
                Disk Name :  EXPLORER
                Free 100    Used  75

Input           File Name : EXP
Input           Program Name : X     - Note: any character will do.
                                             since the Explorer
                                             Auto-Starts
```

FUNCTION and CONTROL KEYS

FCTN KEY

1	**MEMORY WINDOW**	- Toggles the display between the 3 available Memory Windows on the Explorer's Main Screen.
2	**MEMORY SIZE**	- Toggles the display size of the Memory Window on the Explorer's Main Screen through its 4 different sizes.
3	**DISASSEMBLY SIZE**	- Toggles the Next Instruction Display at the bottom of the Explorer's Main Screen between a display of 1 line and 3 lines.
4	**PAGE UP**	- Increases the start address of the current Memory Window displayed by one full page (amount of increase automatically varies with the size of the Memory Window)
5	**SEARCH**	- Activates the search function and allows you to search through a specified address range in the currently displayed memory type, CPU, GROM or VDP (cgv). You can search in Hex, ASCII or ASCII with Basic Bias.
6	**PAGE DOWN**	- Decreases the start address of the current Memory Window by one full page (amount of decrease automatically varies with the size of the memory window)
7	**OPTIONS**	- Activates the Number Converter and Options Screen of the Explorer.
8	**REGISTERS**	- Activates the Gpl/Vdp Status and Vdp Write Only Registers Screen of the Explorer.
9	**EDIT FIELDS/MEM**	- Toggles the cursor between editing of the Explorer's control fields and the Memory Window. (The control fields are above the double line =). On the Number Converter screen FCTN 9 will toggle you between the current field that the cursor is on and the Mathematical and Relational operations selection (ie: Add, Subtract etc.)
0	**BASIC BIAS ON/OFF**	- Toggles the Basic Bias display of the Memory Window on and off which offsets the ASCII display by >60 (96) so you can see the characters as they appear in the Extended Basic and Basic environments. (This does NOT affect the Hex values displayed.)
=	**ASCII/HEX**	- Toggles the Memory Window display between ASCII characters and their Hexadecimal values. If the Basic Bias is on, the ASCII display will be offset by >60.

CONTROL KEYS

CTRL 1	**SINGLE EXECUTION**	- Instructs the Explorer to execute a single instruction, as pointed to by the PC field, and to update the entire Main Screen and Application Program's Screen each time it is pressed and released, according to the instruction executed.
CTRL 2	**CONTINUOUS EXECUTION**	- Instructs the Explorer to continuously execute instructions according to the program flow and to update the Main screen after every instruction (if it is displayed) and the Application Program's Screen until CTRL 2 is pressed again to stop it or until a programmable Break Point is encountered.
CTRL 3	**PRGM/STATUS SCREEN**	- Toggles the current screen display between the Explorer's Main Screen and the Application Program's screen. (note the Explorer executes the program much faster when the actual program screen is displayed)
CTRL 4	**INTERRUPTS ON/OFF**	- Toggles the Interrupt Enable/Disable flag (E D next to the IM field). If the flag is set to E, enable, the Explorer will execute the Interrupt routine(s) each time the Interrupt Mask (IM) does not equal zero (ie: LIMI 1 or LIMI 2 instruction or when you change it yourself)
CTRL 5	**SOUND OFF**	- Pressing and releasing this key will immediately turn off the sound generator and zero out the sound indicator at >83CE in CPU Scratch Pad Ram.
CTRL 9	**SAVE OPTIONS**	- Pressing this key while the Explorer's Options Screen is displayed will write your color and H S options out to the Explorer Disk in drive one. These saved Options will automatically be loaded each time the Explorer is loaded.
CTRL =	**EXIT**	- Pressing this key when the Explorer is NOT in continuous execution mode will EXIT the Explorer to the computer's Title Screen (normal operation is resumed and control is released back to your computer)

SPECIAL KEY

SHIFT **TURBO** - When the Explorer is executing a program in Continuous Execution mode (CTRL 2) with the Explorer's Main Screen displayed you can press the SHIFT key to shift it into TURBO for faster execution of the program. Pressing this key stops the dynamic updating of the entire Main Screen display and only updates the top portion of the screen. Pressing SHIFT and the Enter key, the Space Bar or the CTRL key at the same time will allow the instruction counter to be dynamically updated in TURBO Mode.

A FEW NOTES ABOUT THE KEYS

1. The Explorer executes its CTRL key strokes when you let UP on the key not when it is pressed down. This allows you to press and hold CTRL 2 and then hold down a key that the Application program is looking for and then let up on the CTRL and 2 keys and the Application Program will then accept the other key you are holding down.

2. You can press CTRL 1 and then release the CTRL key but hold down the 1 key and the Explorer will be in auto repeat mode for Single Execution or very slow continuous execution mode. However, if the Application Program goes through the key scan the 1 key WILL BE detected.

3. You can also press CTRL 1 and the Shift keys and then release the CTRL 1 keys and the Explorer will go into Turbo mode until you release the Shift Key.

4. While the Explorer is executing a program (CTRL 2) all of the keys on the keyboard, except CTRL 2 - 5, function normally as far as the Application Program being executed is concerned. (Also see "Key Scan and the Explorer")

5. FCTN QUIT (FCTN =) will only be recognized by the Application Program being executed when the INTERRUPTS are enabled (CTRL 4) since this is part of the level 1 interrupt routine.

STARTING IT UP

TITLE SCREEN

Once the Explorer is loaded you will be greeted by the EXPLORER's Title screen. This screen will only appear when the program is first loaded. At this point you can press ANY key to bring up the Explorer's Main Screen. Once the Main Screen has appeared YOU are in control and ready for some Explorations.

At this time it might be a good idea to play around with the various FCTN keys. You might also want to play around with the Arrow Keys and Enter key to get familiar with cursor's paths through the fields. But, DO NOT change ANY values if you want to continue proper execution of the Application Program. The Explorer has preset these values for you, according to the environment that it was loaded into.

CTRL 3

To see where you left off on the Application Program's screen just press and release CTRL 3. With the Application Program's screen displayed you can press and release CTRL 3 again to bring the Explorer's Main Screen back up. (Note: the Application Program executes Much faster with the Application Program Screen displayed)

CTRL 1 or CTRL 2

With either the Explorer's Main Screen or the Application Program's Screen displayed you can press and release CTRL 1 to execute a single instruction. Pressing and releasing CTRL 2 will turn the Explorer ON and let it continuously execute instructions until CTRL 2 is pressed and released again.

At this point and time just Explore and have fun. YOU CAN NOT HURT YOUR COMPUTER! The worst thing that can happen, if you change some of the values, is that you may lock you your computer and will have to shut it off and reload the Explorer (please see next page for a few Warnings). A little latter on in this manual is a section on Explorations. In this section we will take you step-by-step through various items and document them as we go.

One last point before we leave this page. You may have noticed, if you loaded the Explorer through a running Basic or Extended Basic program that the screen color was set back to Cyan. Unfortunately, there is no way of checking the screen color when the Explorer is loaded so the Explorer sets up default values for the 8 Vdp Write Only registers. If you load the Explorer through a running Basic or Extended Basic program you can easily correct this by placing a CALL SCREEN(x) right after the CALL LOAD("DSK1.xxxxx"). Then when you start up the Explorer it will continue execution of your program and set the screen back to the color you want. Or, you can easily edit the V7 field of the Vdp Register display area and change the screen color.

W A R N I N G W A R N I N G W A R N I N G

W A R N I N G W A R N I N G

DON'TS

Even though the Explorer was made to be as Transparent as possible to the operation of your computer and its peripherals there are a few items you should be aware of.

1. Any TIMING critical operations will NOT function properly. Because the Explorer is operating your computer in Interpretive mode its speed of operation is greatly reduced. This means that ALL Baud rate operations such as RS232 and CS1 or CS2 will not work properly.

2. Since the transfer of data to and from the floppy disk controller is very timing critical these functions will not work properly. The Explorer will allow you to Explore right up to the point where the controller is looking for a device ready from the disk drive but it will not be received so an error is returned. The error is usually 06 - device not ready or no diskette/drive. This will not harm your drives or the floppy contained therein but you will NOT be able to successfully execute OLD, SAVE or CALL LOAD("DSKx.xxxx") type commands.

3. If you own one of the RAM DISK type memory cards, we have found that you CANNOT make ANY Disk type accesses since these DSRs usually take control before the floppy disk controller DSR. When these devices take control they usually do a 32K bank switch and this pages the Explorer out of memory and out of control. Unfortunately they do not return back to us properly since we are executing in Interpretive mode so everything gets locked up. The only thing you can do if this happens is shut down and reload.

4. The Myarc Ram Disk Card (128K Card) also does a Bank switch on Power Up and on EVERY Interrupt. If you own one of these cards you will not be able to successfully go through the Power Up Routine or use the interrupts when the Explorer is in control.

IMPORTANT IMPORTANT IMPORTANT IMPORTANT

5. If you have a Hard Disk Drive hooked up to your computer you can load files through the Explorer since this device is always spinning and in a ready state. HOWEVER, THERE IS A 99.99% CHANCE THAT IT WILL WIPE OUT THE FILE'S HEADER ON THE HARD DISK. We have also had it wipe out a few other files with similar names and quite a number of times it made such a mess of things that the only way to get the hard disk back in operation was to REFORMAT IT!!! Bye Bye files!!!

DO **NOT LOAD FILES OFF THE HARD DISK THROUGH THE EXPLORER UNLESS YOU DON'T CARE IF IT WIPES OUT YOUR HARD DISK!!! - Don't say we didn't warn you!**

EXPLORER'S MAIN SCREEN

```
                      CPU       GROM      VDP       Binary display
                      Control   Control   Control   of Cpu Status
                      Area      Area      Area      Register

                      Cpu       Grom      Vdp        Status
                      ws 83E0   ad 0123   ad 0123    l a e c
                      pc 0024   by   00   by   00    0 0 0 0
                      st 0000   st   00   st   00    o p x im
                      bp FFFF   bp FFFF   bp FFFF    0 0 0 0d  Interrupt mask
                      ------------------------------------    value.
Work Space            r0 0000   r4 0000   r8  0000   r12 0000  e = enabled
Register              r1 0000   r5 0000   r9  0000   r13 0000  d = disabled
values                r2 0000   r6 0000   r10 0000   r14 0000      (CTRL 4)
                      r3 0000   r7 0000   r11 0000   r15 0000

                      v0 00     v2 00     v4  00     v6  00    Vdp Register
                      v1 00     v3 00     v5  00     v7  00    values

MEMORY POINTER        c0000s      Grom Lib 9800      00000000  Instruction
c = Rom or Ram        ====================================    counter in hex.
g = Grom or Gram       00 00 00 00 00 00 00 00 00 00 00 00
v = Vdp Ram            00 00 00 00 00 00 00 00 00 00 00 00    Press FCTN 1 to
0000 = address         00 00 00 00 00 00 00 00 00 00 00 00    display the other
s = Static             00 00 00 00 00 00 00 00 00 00 00 00    2 Memory Windows.
d = Dynamic            00 00 00 00 00 00 00 00 00 00 00 00
      Tracking         00 00 00 00 00 00 00 00 00 00 00 00    FCTN 9 to edit
                      ====================================    Memory or Fields
Next instruction       020D  LI    R13,>9800
to be executed.
```

The UP, DOWN, LEFT and RIGHT arrow keys and the ENTER key will move you from field to field. FCTN 9 will toggle your cursor between editing memory and editing fields. While you are editing the fields or the Memory block in hex you can leave your alpha lock key up or down and the Explorer will automatically adjust your alpha keystrokes into their proper upper or lower case entry. When you are editing the Memory block in ASCII or ASCII with Basic Bias the Explorer will allow you to enter alpha characters in either upper or lower case.

The cursor will not leave a field that is being edited until you press one of the arrow keys or the Enter key. The exception to this is when you press and hold the zero key, this allows you to easily zero out some or all of the fields.

If you press an invalid key or try to input an invalid value, the screen will change to the Error Colors while you are holding down the invalid key. These colors can be set by you on the Explorer's Options Screen.

```
Cpu      ----      ---      ------
ws 83E0  -- ----   -- ----  - - - -
pc 0024  --   --   --   --  - - - -
st 0000  --   --   --   --  - - - --
bp FFFF  -- ----   -- ----  - - - --
```

Cpu Controls

ws - Workspace Pointer

This 2 byte field tracks and sets the Cpu's Workspace Pointer Register. It contains the address of the first register in the currently selected Workspace (main scftware register area for the Cpu). When you or the Application Program change this pointer, the entire Cpu Register display area (R0 thru R15) will also change. For most Grom based software (Basic, X-Basic, Editor/Assembler, Mini Memory and most other cartridges) this Field contains >83E0. For the interrupt routines this field contains >83C0 and for user written Assembly language programs this field can contain any valid even Cpu Ram address. As you change WS the Explorer writes the values in R0 - R15 out to memory. There may be times when you are changing the WS back to 83E0 that the Sound Chip comes on. This happens when the last digit in the WS field is not zero because the Sound Chip resides at >8400. If this happens just press CTRL 5 to turn it Off.

pc - Program Counter

This 2 byte field tracks and sets the Cpu's Program Counter (instruction pointer) Register and it points to the NEXT instruction to be executed. Since the Cpu operates only on even addresses (words not bytes) the least significant bit is always 0. You can place an odd address in this field but it will be rounded down to an even address internally. As this value changes, the disassembly of the next instruction at the bottom of the screen will also change.

st - Status

This 2 byte field tracks and sets the Cpu's Status Register and returns information on the LAST instruction executed that affected the Status Register (see the binary Status display on page 21). The display block in the upper right hand corner of the Main Screen screen (Status) is the binary break down of this register. As this value changes, this binary display area will also change and visa versa.

bp - Break Point

Since the Explorer's powerful machine language interpreter handles all tracking of Cpu memory, Rom and Ram, you can set a Break Point for any valid Cpu PC (program counter) address. Unlike other utility programs this allows you to set Break Points in Rom (Read Only Memory) as well as Ram. When the Cpu's PC register equals the value in the Break Point field the Explorer will halt execution of the Application Program and display the Explorer's Main Screen in the break point colors. Pressing any key will restore the screen to its normal colors and release control to you. (also see Break Points on page 20 for additional information)

```
---      Grom       ---      ------
-- ----  ad 0123    -- ----  - - - -
-- ----  by   00    --    -- - - - -
-- ----  st   00    --    -- - - - --
-- ----  bp FFFF    -- ----  - - - --
```

Grom Controls

ad - Grom Address

This 2 byte field tracks and sets the current Grom address. Since Grom is auto incrementing memory this address will automatically increase by one each time Grom/Gram memory is accessed. The Grom in the 99/4A and modules contain Data and the Graphics Programming Language (GPL) object code which is interpreted and executed by the GPL interpreter in console Rom. Grom is a memory mapped device which means that there are a couple of ports (Cpu memory addresses) that are used to transfer 1 byte at a time to and from Grom/Gram. When an Application Program or the GPL interpreter changes this Grom address it must first write the most significant byte of the Grom address followed by the least significant byte. In the GPL interpreter these address setting instructions may appear something like this:

```
MOVB R6,@>0402(R13)        (move MSB of R6 to >9C02   (>0402+>9800))
MOVB @>83ED,@>0402(R13)    (move LSB of R6 to >9C02   (>0402+>9800))
```

These instructions can be seen when the Cpu PC is at >0060. The Explorer tracks and executes these Grom address writes without any problems. However, if you stop the Application Program in the middle of a Grom Address Write and manually change this field the next instruction will change it again and unpredictable results will occur since the Grom address will most likely be wrong. Do not change the Grom address unless you are experimenting or know where you are setting it to.

by - Byte

This 1 byte field contains the LAST byte read from or written to Grom/Gram. It does NOT contain the byte at the current Grom Address since Grom auto increments its address.(see Memory Windows on page 26 for information on viewing the current byte in Grom memory)

st - GPL Status Byte

This 1 byte field contains a copy of the GPL Status byte which is located in Scratch Pad Ram at >837C. When you or the Application Program change the byte at >837C this field will also change and visa versa. A binary break down of the bits in this Status byte can be seen and edited by pressing FCTN 8 - Registers. (see Registers Screen on page 45)

bp - Grom Break Point

Since the Explorer's powerful machine language interpreter handles all tracking of Grom you can set a Break Point for any valid Grom address. When the Grom address (AD) equals the value in the Break Point field the Explorer will halt execution of the Application Program and display the Explorer's Main Screen in the break point colors. Pressing any key will restore the screen to its normal colors and release control to you. (also see Break Points on page 20 for more information)

```
---    ----    Vdp       ------
-- ---- -- ----  ad 0123   - - - -
-- ---- --   --  by   00   - - - -
-- ---- --   --  st   00   - - - --
-- ---- -- ----  bp FFFF   - - - --
```

Vdp Controls

ad - Vdp Address

This 2 byte field contains the current Vdp read/write address or write only Vdp register being accessed. The specific Vdp operation taking place is determined by the value written to >8C02, the port (Cpu memory address) for the Vdp processors Write Address Register which is reflected in this field.

```
>0000 - >3FFF = a Read from Vdp Memory operation
>4000 - >7FFF = a Write to Vdp Memory operation (address + >4000)
>80xx - >87xx = a Write to Vdp Register Operation
                      (ie: >81E0 sets Vdp register 1 to >E0)
                      (    >8320 sets Vdp register 3 to >20)
```

Because the Vdp memory address is auto incremented by the Vdp Processor this address will increase by one after each Vdp read or write access. Once again since Vdp is a memory mapped device there are a couple of ports (Cpu memory addresses) that are used to transfer 1 byte at a time to and from Vdp memory and the Vdp Chip. When an Application Program changes this Vdp address it must first write the least significant byte of the Vdp address followed by the most significant byte. These address setting instructions may appear something like this:

```
SWPB R6              (set up least significant byte)
MOVB R6,*R15         (move LSB of R6 to >8C02)
SWPB R6              (set up most significant byte)
MOVB R6,*R15         (move MSB of R6 to >8C02)
```

The Explorer tracks and executes these set Vdp Address operations without any problems. IMPORTANT - Do Not change the PC, WS or Vdp AD fields in the middle of a set Vdp Address operation. This will cause future set Vdp Address operations to be out of sync until a Vdp Read data is executed, which will reset the Vdp chip and the Explorer to the proper state.

by - Byte

This 1 byte field contains the Last byte read from or written to Vdp Ram. It DOES NOT contain the byte at the current Vdp Address (AD) since Vdp auto increments its address after each read or write. (see Memory Windows on page 26 for information on viewing the current byte in Vdp memory)

```
---     ----      Vdp      ------
-- ---- -- ----   -- ----  - - - -
-- ---- --   --   --   --  - - - -
-- ---- --   --   st   00  - - - --
-- ---- -- ----   bp FFFF  - - - --
```

Vdp Controls continued

st - Vdp Status

This field contains a copy of the Vdp Status register when the Application Program screen was last displayed. This field is not updated while the Explorer's Main Screen is up, since it would just be displaying the Vdp Status of the Explorer and not the Application Program. By toggling screens (CTRL 3) from time to time you can update this field. You can not edit the value in this field since this Vdp register is a READ ONLY register (except to place zero's in it).

bp - Break Point

Once again the Explorer's powerful machine language interpreter handles all tracking of Vdp Memory. This adds a powerful function to the Explorer which allows you to set some very specific break points for Vdp accesses.

Break Point for a specified READ FROM VDP ADDRESS - >0000 thru >3FFF
ie: BP 02E0 - halts Explorer on Vdp Read at address >02E0

Break Point for a specified WRITE TO VDP ADDRESS = >4000 thru >3FFF
ie: BP 42E0 - halts Explorer on Vdp Write at address >02E0

Break Point for a specified WRITE TO VDP REGISTER = >80xx thru >87xx
ie: BP 81E0 - halts Explorer when Vdp register 1 gets set to >E0

When the Vdp address (AD) equals the value in the Break Point field the Explorer will halt execution of the Application Program and display the Explorer's Main Screen in the break point colors. Pressing any key will restore the screen to its normal colors and release control to you. (also see Break Points on the next page for more information)

```
Cpu       Grom      Vdp       ------
-- ----   -- ----   -- ----   - - - -
-- ----   --   --   --   --   - - - -
-- ----   --   --   --   --   - - - --
bp 02B2   >> 6000   bp 4300   - - - --
```

Programmable Break Points

Since you are in complete control of the Application Program you can stop and start it at will. This could be considered as unlimited break points. However, the Explorer will allow you to program up to three different specific Break Points, one for each type of memory.

To program a Break Point, move the cursor to the BP field for any one of the memory types and type in the address you would like the Explorer to halt on. These Break Points will halt the Explorer when it is in Continuous Execution Mode (CTRL 2), no matter which screen is displayed. If the Application Program's screen is displayed, the Explorer will halt execution and bring up the Main Screen. The screen will be displayed in the Break Point colors with two greater than signs (>>) pointing to the type of memory that caused the Break Point. At this time the Explorer is waiting for you to press a key and acknowledge the Break point condition. Once you press a key the normal Main screen colors will be displayed and the two >> signs will be replaced with the letters BP.

The Explorer was designed two allow you to use the Single Execution key (CTRL 1) to get past a break point. This also means that when you are Single Executing (CTRL 1) a program, the Break Points will not be activated and the screen will not change to the Break Point colors.

To turn OFF the Break Point(s) simply place a value in this field that should never be reached by that type of memory. We have found that FFFF works very well for all types of memory. The Cpu's PC should never be equal to FFFF since this is an odd address and it is also part of the Load interrupt vector. Since Grom is a byte oriented memory mapped device this could be a valid address but most of the modules that we have seen only have valid Grom addresses up to F7FF. The exception to this rule is a new German Extended Basic module which has a data table that goes up to FFFF when you execute CALL APESOFT. In Vdp memory the highest address set should never be above 87FF since this is the highest valid Write to Vdp Register value.

```
---      ----      ---      Status
-- ----  -- ----   -- ----  l a e c
-- ----  --   --   --   --  1 0 1 0
-- ----  --   --   --   --  o p x im
-- ----  -- ----   -- ----  0 0 1 0e
```

Cpu Status

These fields contain the binary representation of the Cpu's Status (ST) register. Not all of the 16 bits in this register are active so only the active bits are displayed here. These bits can be thought of as binary On (1) and Off (0) switches. Also the interrupt mask field can contain hex values in the range of 0-F to indicate the current Hardware interrupt level allowed.

Active bits in the 9900 Cpu's Status Register:

```
L> A> EQ C  OV OP X                 Int Mask
0  0  0  0  0  0  0 nu nu nu nu nu 0 0 0 0
```

l - **Logical Greater Than (L>)**
When this bit is set, equal to 1, it indicates that the last instruction executed that effected this bit in the Cpu's Status register resulted in a logical (unsigned number) greater than condition.

a - **Arithmetic Greater Than (A>)**
When this bit is set it indicates that the last instruction executed that effected this bit in the Cpu's Status register resulted in an arithmetic (signed number) greater than condition.

e - **Equal (EQ)**
When this bit is set it indicates that the last instruction executed that effected this bit in the Cpu's Status register resulted with the words or bytes operated upon being Equal or the word or byte being zero.

c - **Carry (C)**
When this bit is set it indicates that the last instruction executed that effected this bit in the Cpu's Status register caused the most significant bit of the word or byte operated upon to be carried out of the operand into this bit in the Status register.

o - **Overflow (OV)**
When this bit is set it indicates the the last instruction executed that effected this bit in the Cpu's Status register resulted in a too large or too small condition for signed numbers.

p - **Odd Parity (OP)**
This bit is used in Byte operations. When set it indicates that the that the parity of the destination byte operand of the last instruction executed that effected this bit in the Cpu's Status register has an odd parity. Odd parity means that the number of bits that are on in the byte add up to an odd value. Example: 01001100 (3 bits on) is odd parity and 01001101 (4 bits on) is even parity.

```
---       ----       ---      Status
-- ----  -- ----   -- ----   - - - -
-- ----  --   --   --   --   - - - -
-- ----  --   --   --   --   - - x im
-- ----  -- ----   -- ----   - - 1 0e
```

Cpu Status continued

x - Extended Operation (X)

When this bit is set it indicates that that the Application Program flow has been transferred to one of the XOP vectors to continue execution. XOPs are Software controlled interrupts for the 9900 microprocessor. On the 99/4A only XOPs 0, 1 and 2 are implemented and on some 4A's only XOPs 0 and 2 are. XOP vectors are located in Cpu Rom starting at >0040 for level 0.

```
Level 0    >0040  = >280A - Workspace for unreleased Debugger Card
           >0042  = >0C1C - Program Counter (PC) for XOP 0

Level 1    >0044  = >FFD8 - Workspace (WS) for XOP 1 (if implemented)
           >0046  = >FFF8 - Program Counter (PC) for XOP 1

Level 2    >0048  = >83A0 - Workspace (WS) for XOP 2
           >004A  = >8300 - Program Counter (PC) for XOP 2
```

Levels 3 thru 15 are not implemented on the 99/4A.

im - Interrupt Mask Value

The hex value in this field represents the least significant 4 bits of the Cpu's Status register. These bits set the highest level of Hardware interrupt allowed by the 9900 microprocessor. On the 99/4A an interrupt can be triggered by an external peripheral device, the Vdp vertical retrace (60 times a second) or the clock on the 9901 for CS1 & CS2. The 99/4A has only levels 0 & 1 and the non-maskable Load interrupt implemented. Level 0 is the reset interrupt (power up routine). Level 1 controls the Cassette Timing, Peripheral Interrupt Routines, Auto Sprite Motion, Auto Sound Processing and the Quit Key. Level 1 also executes the User Interrupt Routine pointed to by the address in >83C4, if it does not contain zero. The vectors that are in the level 2 position in Rom are for a routine that blanks the screen after a certain amount of inactive time, this is not an actual Hardware Interrupt Level. The interrupt vectors are located in Cpu Rom memory starting at >0000.

```
Level 0      >0000 = >83E0 - Workspace (WS) for Reset (Power Up routine)
             >0002 = >0024 - Program Counter (PC) for Reset

Level 1      >0004 = >83C0 - Workspace (WS) for Level 1 Interrupt
             >0006 = >0900 - Program Counter for Level 1 (most consoles)
```

Hardware Levels 2 thru 15 are not implemented on the 99/4A.

```
---      ----       ---      Status
-- ----  -- ----    -- ----  - - - -
-- ----  --   --    --   --  - - - -
-- ----  --   --    --   --  - - - im
-- ----  -- ----    -- ----  - - - 0e
```

Cpu Status (Interrupt Mask) continued

Unlike any other utility program currently available for the 99/4A, the Explorer will execute interrupts during the normal flow of the Application Program. Whenever the Application Program or you set the IM field to a value that is greater than zero, (LIMI 1 thru 15 instruction) and interrupts are enabled (CTRL 4) the Explorer will stop execution of the current Application Program and begin execution of the interrupt routine pointed to by the level 1 vectors. After completing the interrupt routine the Explorer will continue execution of the current Application Program where it left off.

To enable or disable interrupt execution press CTRL 4. The E or D indicator after the im hex value will change accordingly.

Examples:

```
im
0E  = interrupt execution enabled

im
0D  = interrupt execution disabled
```

NOTE

When interrupts are enabled and the Explorer is running with the Application Program's screen displayed, you may notice that the Peripheral card lights will blink on and off. This happens because the interrupt routine is searching through the peripherals for their interrupt routines. When the Explorer is running with the Explorer's Main Screen displayed, the screen interrupt takes control before the peripheral scan so these lights will not come on. Also, when the interrupts are enabled the Application Program executes slower because of the extra code in the interrupt routine. It is not necessary to ALWAYS have the interrupts enabled for proper execution of the Application Program.

r0 0000	r4 0000	r8 0000	r12 0000
r1 0000	r5 0000	r9 0000	r13 0000
r2 0000	r6 0000	r10 0000	r14 0000
r3 0000	r7 0000	r11 0000	r15 0000

Workspace Registers

r0 - r15 - Cpu's Workspace Registers

This portion of the Explorer's Main Screen contains a copy of the Application Program's current Workspace Registers. When the Explorer's Main Screen is displayed these registers are automatically updated on the screen after each and every instruction is executed. Any changes that you make to these registers or the actual area of memory pointed to by the Cpu WS field will affect the Application Program's workspace.

Generally speaking the workspace registers can contain most anything that the programmer wants. However, there are a few registers that must, or will, contain certain items for, or after, the execution of some instructions.

r0 - Holds shift count for a some of the Shift instructions

r11 - Stores the Return address for the Branch and Link (BL) instruction and the Effective Address after an XOP.

r12 - Holds the CRU base address during cru bit access instructions (SBO, SBZ, TB, LDCR, STCR)

r13 - Stores the old Workspace Register (WS) value after a context switch, like the BLWP or XOP instruction or the execution of the interrupt routines.

r14 - Stores the old Program Counter Register (PC) value after a context switch.

r15 - Stores the old Cpu Status Register (ST) value after a context switch.

(see the Scratch Pad Ram Memory Map in the Appendixes for additional information on the Interrupt workspace at >83C0 and the GPL workspace at >83E0)

v0 00	v2 F0	v4 F8	v6 F8
v1 E0	v3 0C	v5 86	v7 07

Vdp Registers

These eight 1 byte fields track the TMS 9918A Video Display Processors's eight Write Only Registers. These Write Only Registers are used by the TMS 9918A to set up the various Vdp modes, graphics, text, bit map etc., and the various table locations, screen table, character table, sprite attribute table etc.

v0 - Bit Map mode & External Vdp Chip enable

v1 - 4K/16K Vdp Ram, Screen Enable/Disable, Vdp retrace Interrupt, Text mode, Multi-Color mode, Sprite Size & Sprite Magnification

v2 - Screen Image Table base address - (times >400)

v3 - Color Table base address - (times >40)

v4 - Character Pattern Table base address - (times >800)

v5 - Sprite Attribute Table base address - (times >80)

v6 - Sprite Pattern Table base address - (times >800)

v7 - Text mode foreground color (most significant nibble) and the Screen color for all modes (least significant nibble)

You can edit any one of these fields and then press CTRL 3 to bring up the Application Program's screen and immediately see what effect your edit has had on the Application Program.

Pressing FCTN 8 - Registers will display these registers with their binary break down and the tables multiplied out to their proper base addresses.

(Also see Registers Screen for more information on these registers)

```
Memory Pointer —— c0000s     ---- --- ----    --------
                          ==================================
                          00 00 00 00 00 00 00 00 00 00 00 00
                          xx 00 00 00 00 00 00 00 00 00 00 00
                          00 00 00 00 00 00 00 00 00 00 00 00   > Memory Window
                          00 00 00 00 00 00 00 00 00 00 00 00
                          00 00 00 00 00 00 00 00 00 00 00 00
                          00 00 00 00 00 00 00 00 00 00 00 00
                          ==================================
```

Memory Windows

The three Memory Windows are a very powerful part of the Explorer. These Windows can be set to any valid address for the specified type of memory (Cpu, Vdp, Grom/Gram) either by you or the Application Program. Pressing FCTN 1 will toggle the Memory Window display between the three Memory Windows. These Windows can be any combination of Cpu, Grom or Vdp memory types as well as static or dynamically tracked. Using the Memory Edit function of the Explorer allows you to easily examine (any area) and change (Ram, Gram) areas of memory as you wish. These areas of memory can be displayed in Hexadecimal, ASCII or ASCII with the Basic Bias. The Memory Window has a Memory Pointer and Mode Indicator above it that can be edited by you. This Memory Pointer will also update itself as you move the cursor around in the Memory Window to indicate the exact address that the cursor is sitting on.

c0000s - Memory Window Pointer

This pointer is actually three different fields that consist of the following items:

c-----

This indicates which type of memory is displayed in the memory block. You can place any one of the following three alpha characters in this field and the memory block will instantly update itself and display that type of memory:

c = Cpu Memory (Rom/Ram)
g = Grom/Gram Memory
v = Vdp Ram Memory

-0000-

When the cursor is NOT in the memory block or when the Explorer is in Continuous Execution mode, this indicates the current start address for the memory block displayed. When the cursor is in the memory block this indicates the exact address that the cursor is sitting on. The valid ranges for the different types of memory are as follows:

>0000 - FFFF for Cpu Memory
>0000 - FFFF for Grom Memory
>0000 - 3FFF for Vdp Memory

```
-----s      ---- --- ----      --------
========================================
00 00 00 00 00 00 00 00 00 00 00 00
xx 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
========================================
```

Memory Windows continued

-----s
This indicates the MODE for the memory block. The Static mode (S) will leave the memory block exactly where you place it. The Dynamic (D) mode allows the memory block to follow the Application Program's access to the type of memory displayed as follows:

c----d
If the Type indicator is a C (Cpu Rom/Ram memory) and the mode indicator is a D (dynamic) the memory block will automatically follow the Cpu's PC register.

g----d
If the Type indicator is a G (Grom/Gram memory) and the mode indicator is a D (dynamic) the memory block will automatically follow the Grom Address (AD).

v----d
If the Type indicator is a V (Vdp Ram memory) and the mode indicator is a D (dynamic) the memory block will automatically follow the Vdp Address (AD).

NOTE: When the displayed Memory Window Mode is Dynamic the address that is being tracked starts on the second row of the Memory Window. This allows you to see the 12 bytes prior to the current tracked address.

```
                        -----d      ---- --- ----      --------
                        ========================================
                        00 00 00 00 00 00 00 00 00 00 00 00
Actual byte at -------- xx 00 00 00 00 00 00 00 00 00 00 00
tracked address         00 00 00 00 00 00 00 00 00 00 00 00
```

To change the Mode for the displayed Memory Window simply place an S or D in this field. Each one of the three Windows can have their own Mode.

FCTN 1 **FCTN 2** **FCTN 3** **FCTN 4**

FCTN 6 **FCTN 9** **FCTN 0** **FCTN =**

Memory Window & Memory Editor Keys

The Following Keys Effect The Memory Window Display

FCTN 1 - Toggles the display to one of the 3 different Memory Windows.

FCTN 2 - Changes the size of the Memory Window display to one of the 4 main sizes.

FCTN 3 - Changes the size of the Next Instruction display area which affects the size of the Memory Window display.

FCTN 4 - Increases the start address of the Memory Window by one full Page (Window). The amount of increase automatically compensates for the various Memory Window sizes.

FCTN 6 - Decreases the start address of the Memory Window display by one full page (Window). The amount of decrease automatically compensates for the various Memory Window sizes.

FCTN 9 - Toggles you in and out of the Edit Memory Mode. Or, to put it another way, it toggles you between editing of memory and editing of fields.

FCTN 0 - Toggles the Basic Bias On and Off. When this Bias is On, the words "BASIC BIAS" will be displayed in the middle of the top double line (=). The bias ONLY effects the ASCII display of the Memory Window by offsetting the ASCII display with >60 to match the Basic environments. This Bias is most useful for editing the Screen Image area of Vdp Ram when you are in the Basic or Extended Basic environments. With this Bias you can also see the various ERROR messages in Grom for the two Basic languages.

FCTN = - Toggles the Memory Window display between Hexadecimal values and their ASCII characters.

```
-0000-      ---- --- ----     --------
======================================
 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00
======================================
```

Memory Editor

The Explorer contains a very powerful and easy to use Memory Editor. To enter the Memory Editor when the cursor is in one of the fields at the top portion of the screen (above the double line =) just press FCTN 9. To exit the Memory Editor (when the cursor is in the Memory Window) you also press FCTN 9. The FCTN 9 key on the Main screen toggles you between editing the fields and editing memory.

This Memory Editor allows you to examine any area and type of memory in your computer system. It was designed with future expansion in mind in that it treats ALL areas of Cpu memory as if it is Ram and ALL areas of Grom memory as if it is Gram (Auto Incrementing Graphics Ram the complement to Grom). By this we mean that the Explorer will allow you to attempt to write to any area of memory. After you have typed in the value or character into a memory location the Explorer attempts to write that value to the memory location and then rereads and redisplays the current Memory Window. If the location was Ram or Gram the Memory Window will reflect the change, if the location was Rom or Grom the change will not be displayed. With this feature any future Ram or Gram additions or modifications to your system can easily be edited. (Note: There currently isn't any GRAM in our systems)

The size of the Memory Window/Edit Area can be changed at any time by simply pressing FCTN 2. There are 4 main sizes available. So, when you have reached the largest size (full screen) and you press FCTN 1 again the Memory Window will automatically drop back down to the smallest size.

The Type of memory displayed (Cpu, Grom, Vdp) can be changed at any time by pressing FCTN 1 which brings in the next Memory Window you have set. Or, you can leave the Memory Edit mode and place the cursor at the start of the Memory Pointer field (c0000s) and input a C,G or V there.
NOTE: If your Windows are in Dynamic (D) Mode pressing FCTN 1 will display the area of memory that is being tracked. If you do not want the address to change as you toggle between Memory Windows (FCTN 1) simply change them to Static (S) Mode.

UP **DOWN** **LEFT** **RIGHT**

SHIFT UP **SHIFT DOWN** **SHIFT LEFT** **SHIFT RIGHT**

Memory Editor continued

The Explorer's Memory Editor is a FULL SCREEN type of editor. This allows you to move the cursor around in the Memory Window with the four arrows keys (FCTN Up, Down, Left & Right arrows). As you move the cursor around in the Memory Window the address in the Memory Pointer (c0000s) will update itself and indicate the exact address that the cursor is currently sitting on. Also ALL of the keys in the Memory Editor will auto repeat if you hold them down.

Pressing the UP arrow key when the cursor is sitting on the Top row of the Memory Window will allow you to scroll the Memory Window address and display down 12 bytes (1 row). Conversely, pressing the DOWN arrow key when the cursor is on the Bottom row of the Memory Window will allow you to scroll the Memory Window address and display up 12 bytes (1 row).

Pressing the LEFT arrow key when the cursor is sitting in the upper left hand corner (first byte of the Memory Window) will allow you to scroll the Memory Window start address and display down 1 byte. Conversely, pressing the RIGHT arrow key when the cursor is sitting in the lower right hand corner (last byte of the Memory Window) will allow to scroll the Memory Window address and display up 1 byte.

The Explorer also has a unique feature in its Memory Editor in that you can lock the cursor on a given byte and drag the memory display around in the Window boundaries. To lock the cursor on a given byte (address) just press the FCTN and SHIFT keys down at the same time as you press the arrow keys. This allows you to drag a given byte to any location in the Window. This is very useful for positioning a given byte in the home (upper right hand corner) position of the Window.

We would like to recommend at this time that you play with the various FCTN, Arrow and Shift keys of the Explorer's Memory Window and Memory Editor. This will help you to become more familiar and comfortable with its powerful features and to make future Explorations easier.

```
                  c0000-   st 0000   fn 0000   --------
                  ==========================================
Search String ——— 00 00 00 00 00 00 00 00 00 00 00 00
                  ------------------------------------------
                  00 00 00 00 00 00 00 00 00 00 00 00\
                  00 00 00 00 00 00 00 00 00 00 00 00 >Memory display
                  00 00 00 00 00 00 00 00 00 00 00 00/
                  00 00 00 00 00 00 00 00 00 00 00 00
                  ==========================================
```

Search Function

The Explorer also contains a very powerful and easy to use Search Function. This function allows you to search through any type of memory, Cpu, Grom or Vdp, in Hexadecimal, ASCII or ASCII with Basic Bias. It also allows you to search forwards, low to high address range, or backwards, high to low address range. To activate the Search Function simply follow these steps:

1. Select the type of memory you wish to search through by placing a C, G or V in the Memory Pointer field or by pressing FCTN 1 until the proper type of memory is displayed. You must do this PRIOR to activating the Search Function.

2. Press FCTN 5 - Search - to activate the Search Function.

3. Input the Start Address in the ST (Start) field and the End Address in the FN (Finish) field and then press Enter or the Down Arrow.

4. Select either Hex display for a Hexadecimal search or ASCII display for a ASCII search by pressing FCTN = - ASCII/HEX Toggle.

5. If you have selected ASCII, you can select either normal ASCII or ASCII with the Basic Bias by pressing FCTN 0 - Basic Bias On/Off.

6. Type in the Hex value(s) or ASCII letters, according to the display, and then move the cursor back one space (left arrow key) to place it ON TOP OF THE LAST CHARACTER OR BYTE in your search string, and now press ENTER to start the search.

Example:

```
g0000-   st 2000   fn 57FF   --------
=====================================
T I   B A S I [C] . . . .
-------------------------------------
```

Pressing Enter with the cursor sitting ON TOP OF the C in 'BASIC' will instruct the Explorer to search through Grom (G) memory from >2000 through >57FF for the first occurrence of 'TI BASIC' and to display its start location in the Memory Window and Memory Pointer. If you place your cursor on top of the I in 'TI' the Explorer will ignore the rest of the search string and search for the first occurrence of 'TI' in the selected Grom memory range.

```
g2152-    st 2153   fn 57FF   --------
======================================
T I   B A S I [C] .  .  .  .
--------------------------------------
T I   B A S I  C  B  +  .  ¦
| | | | | |  .  .  .  .  .  .  .
. . . . . .  s  .  .  '  .  1
. . . .   .  .  .  .  .  @  .
======================================
```

Search Function Continued

When the search string is found the Explorer will automatically display that area of memory in the Memory Window. It will also update the Memory Pointer to the address where your search string was found. It also adjusts the Search Function Start Address (ST) to allow you to just press Enter again to search for the next occurrence. If you are searching forward through memory it will set the Start Address (ST) to one address higher than the Memory Pointer value. If you are searching backwards through memory it will set the Start Address (ST) to one address lower than the Memory Pointer value.

If the search string is not found the Explorer will NOT update the Memory Pointer, Memory Window or Start Address. If the first line of the Memory Window, up to and including the cursor location, does not match the search string display then your string was NOT found.

To change the ST or FN fields when the cursor is in the search string area just press the Up arrow key. When you are finished with the Search Function just press FCTN 5 again to exit it. NOTE: This is the ONLY way to leave the Search Function and to resume normal operation of the Explorer.

When you exit the Search Function the Explorer will remember the search string you input and the Start and Finish addresses where it left off. So, the next time you enter the Search function these items will automatically be displayed. However, it does not change the current memory type to the one you searched through before. This allows you to easily search through Cpu, Grom and Vdp for the same item(s).

NOTE

The Search Function is an independent function of the Explorer. When it is activated most of the other functions of the Explorer cannot be accessed (ie: FCTN 7 Options, CTRL 2 Continuous Execution etc.). Trying to activate one of these functions will cause the screen to change to its Error colors, which indicates that you need to leave the Search Function (press FCTN 5 again) before accessing this other function.

```
------      Grom Lib 9800     00000000
=========================================
-- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- --
=========================================
```

Grom Library & Instruction Counter

Grom Lib 9800 - Grom Library Page

This field was added to the Explorer for two reasons. First to allow you to watch the 4A's operating search through the Grom Library for a particular CALL. And secondly for future expansion of your computer system. There currently isn't a Grom Library Expansion box available for your computer but there may be one in the near future and the Explorer is fully set up to handle it and GRAM simulators.

You may have noticed at one time or another, when you removed or inserted a module that the REVIEW MODULE LIBRARY message came up in your menu. And, you probably have tried to make that selection to see what happens. Unfortunately, since there wasn't a Grom Library Expansion box hooked up, nothing happened. You can, however, watch TI Basic attempt to search through the nonexistent Grom Library. To do this simply load the Explorer into the Basic environment and start it up. When the cursor reappears and you are in Command/Edit mode of Basic slowly type in an invalid CALL such as, CALL MG and press Enter. As the computer searches for this invalid CALL you will notice that this field is updated to each of the Library pages until Basic can not find the CALL and returns the * BAD NAME error message. (There will be more documentation on this when a Grom Library box is available)

00000000 - Instruction Counter

This four Byte (eight nibble or 32 bit) field is the Instruction Counter in hexadecimal that counts up to >FFFFFFFF before it resets itself. You can edit this field at any time and set it to any value or reset it back to zero. This field increments itself by one each time the Explorer executes a 9900 Machine Language Instruction. It is displayed on the Main Screen in hexadecimal, for purposes of speed, and it is also displayed on the Options screen in Decimal format. This way you can simply press FCTN 7 at any time to see the actual number of instructions (in decimal) executed so far. It is also handy as a quick hexadecimal to decimal number converter for converting large hex numbers into their decimal value. >FFFFFFFF equals 4,294,967,295 which is also the address range of a true 32 bit computer - 4 Gigabytes.

```
DB46  MOVB R6,@>0402(R13)
```

— or —

```
Object       DB46  MOVB R6,@>0402(R13)            Source
 Code        0402                                  Code
             DB60  MOVB @>83ED,@>0402(R13)
```

Instruction Disassembly

The last display area on the Explorer's Main screen is the Next Instruction to be executed. This area can be displayed in two sizes, single line or three lines. The single line display will always display the complete Next instruction, no matter how many object code words make up the instruction. The three line display can display up to three instructions at a time if the first two instruction are only one word of object code in length.

As each instruction is executed this area is updated and the next instruction to be executed will be displayed. This display area dynamically follows the Cpu's PC register and the area of memory that the PC points to.

If you place the PC field on an area of Cpu Ram, say >2900, and then place the Memory Window at c2900s, you can edit the values in the Memory Window and watch the disassembly display change to the new instruction.

```
EXAMPLE:          Cpu
     Input ---- ws 2800
     Input ---- pc 2900
                st ---- = any value is ok
                bp FFFF

     Set ------ c2900s     Grom Lib 98--    --------

     Type in ---04 20 00 00 -- -- -- -- -- -- -- --
                -- -- -- -- -- -- -- -- -- -- -- --
     which
     equals ----0420  BLWP @>0000
```

By inputting the values indicated and typing in >04 20 00 00 into the Memory Window you will set the next instruction to do a Branch and Load Workspace Pointer @>0000. This will begin execution of the Power Up routine when you press CTRL 1 or CTRL 2. Go ahead and press CTRL 1 and note R13, R14 and R15 of the new Workspace.

EXPLORER'S OPTIONS SCREEN

```
---------- Number Converter ----------

hex  FFFF  + 0000    =  0000FFFF

dec  65535 + 00000   = 0000065535

bin  11111111 11111111     car 0  ovr 0
   + 00000000 00000000     + add  a and
                           - sub  o or
   = 00000000 00000000     * mul  x xor
     11111111 11111111     / div  n not

---------- Explorer Options ----------

Cru Base     1100  0

Load Characters    h       s step speed
Execute Key Scan   h       h high speed

Colors          text/scrn    Counter
 Status Screen     F4
 Break Point       1A      d00000000000
 Error Condition   F6
```

The Explorer's Option Screen actually contains four sections, the Number Converter, The Cru Base switch, the Options section and the decimal Counter. The Number Converter has the same mathematical logic as the 9900 Microprocessor. The Cru Base switch actually performs two functions. First it allows you to turn On (1) and Off (0) different Cru bits. Secondly it Reads back the selected Cru Base (bit) and displays it. The Options section of this screen allows you to configure the Explorer to your tastes and to save these Options on the Explorer Disk.

To activate this screen just press FCTN 7 - Options - from either of the other two Explorer Screens (Main or Registers), while the Explorer is NOT Continuously Executing (CTRL 2) an Application Program. Once you have activated this screen you can return to the Explorer's Main Screen by pressing FCTN 7 again.

```
                ---------- Number Converter ----------

Operation        hex  FFFF  + 0000    =  0000FFFF
Indicator
                 dec  65535 + 00000   = 0000065535

                 bin  11111111 11111111     car 0  ovr 0
                    + 00000000 00000000     + add  a and      Operation
                                            - sub  o or       Selection
                    = 00000000 00000000     * mul  x xor      Fields
                      11111111 11111111     / div  n not
```

Number Converter

The Explorer's Number Converter will convert Hexadecimal, Decimal and Binary numbers and perform the mathematical and relational operations of Add, Subtract, Multiply, Divide, AND, OR, XOR and NOT. The valid range for the different number bases is >0000 thru >FFFF, 0 thru 65535 and 0000000000000000 thru 1111111111111111. This Number Converter operates dynamically in that ALL of the Number Converter fields are automatically updated as you type in EACH character, so type slow.

When this screen is active the action performed by the the FCTN 9 key is changed. Pressing FCTN 9 takes your cursor from wherever it is on the screen and places it on the + sign in the mathematical and relational selection fields. When the cursor is already on one of these fields, pressing FCTN 9 will place the cursor back to the field that it was in.

The mathematical and relational operations have the same logic as the 9900 microprocessor. What this means is that they calculate the same result as the corresponding 9900 instruction. All of the mathematical and All of the relational operations except NOT operate on two numbers. The NOT relation works on a single number and the Explorer is set up to NOT a value that is in the input field following the operation indicator.

The CAR 0 and OVR 0 indicators above the mathematical and relational selection fields simulate the CARRY and OVERFLOW bits of the Cpu's Status Register. These bits will be set or cleared on the various operations according to the operation performed and its result.

Number Converter Continued

Number Conversion

To perform just a straight number conversion simply type in the number into the appropriate number base field. The Explorer will dynamically convert the number as you type it in. The proper conversion will be displayed in the the other two number bases's corresponding input fields. By looking in these fields for the result instead of the fields following the = sign you will not have to worry about the mathematical or relational operation indicator.

Example - Convert >2006 into Dec and Bin

```
Input          hex  0000   + 2006    =  00002006

Result         dec  00000 + 08198   = 0000008198

               bin  00000000 00000000    car 0  ovr 0
Result            + 00100000 00000110    + add  a and
                                         - sub  o or
                  = 00000000 00000000    * mul  x xor
                    00100000 00000110    / div  n not
```

Mathematical Operations

To perform a mathematical operation (add, sub, mul or div) input the two values into their proper number base input fields and press FCTN 9. With the cursor sitting on the + sign in the mathematical and relational selection fields you can move it around to the proper operation. When the cursor is on the proper operation simply press ENTER and that operation will be performed. Once an operation is selected the Explorer automatically executes it whenever you input values.

Example - Divide 16500 by 8198

```
                                     Quotient
               hex  4074  / 2006    =  00020068
                                                 Remainder
Input          dec  16500 / 08198   = 0000200104

               bin  01000000 01110100    car 0  ovr 0
                  / 00100000 00000110    + add  a and
                                         - sub  o or
Quotient ------ = 00000000 00000010     * mul  x xor
Remainder ------- 00100000 01101000     / div  n not    Select
```

So 8198 goes into 16500 2 times with a remainder of 104.

Number Converter Continued

Relational Operations

Performing a relational operation (AND, OR, XOR or NOT) is much the same as performing a mathematical operation. Input the two values into their proper number base input fields and press FCTN 9. With the cursor sitting on the + sign in the mathematical and relational selection fields you can move it around to the proper relational operation. When the cursor is on the proper operation simply press ENTER and that operation will be performed.

Example - Perform >42E0 AND >3FFF

```
Input        hex  42E0  a 3FFF   =  000002E0  ---- Result

             dec  17120 a 16383  = 0000000736 ---- Result

             bin  01000010 11100000   car 0  ovr 0
                a 00111111 11111111   + add  a and    Select
                                      - sub  o or
                = 00000000 00000000   * mul  x xor
Result  -------   00000010 11100000   / div  n not
```

So this AND masks off the two high order bits and leaves us with >02E0

The NOT operation is a single number operation. So the Explorer expects the value for a NOT to be in the input fields AFTER the Operation Indicator

Example - NOT >BCDE

```
Input        hex  ----  n BCDE   =  00004321  ---- Result

             dec  ----- n 48350  = 0000017185 ---- Result

             bin  -------- --------   car 0  ovr 0
                n 10111100 11011110   + add  a and
                                      - sub  o or
                = 00000000 00000000   * mul  x xor
Result  -------   01000011 00100001   / div  n not    Select
```

So NOT turns Off the bits that are On, and On the bits that are Off

Number Converter Continued

Hexadecimal to CALL PEEK or CALL LOAD

Converting Hex numbers into their Decimal CALL PEEK or CALL LOAD addresses is a simple matter if you keep in mind the following rule. IF the Hex number is greater than >8000 then subtract it from Zero and look at the result. IF it is less than >8000 just do a straight number conversion. In this example the Hex number is greater than >8000 so we will subtract it from Zero. To do this input four zero's in the first Hex field, input 83E0 in the second Hex field, press FCTN 9, move the cursor down to - (sub) and press ENTER.

Example - Convert >83E0 into its Decimal CALL PEEK address.

```
Input        hex  0000  - 83E0   =  00007C20

             dec  00000 - 33760  = 0000031776 ——— Result
```

So >83E0 can be peeked into with CALL PEEK(-31776,A,B)

CALL PEEK or CALL LOAD to Hexadecimal

Converting CALL PEEK or CALL LOAD values into their Hex locations is very similar to the above conversion. The rule to keep in mind here is - IF the CALL PEEK or CALL LOAD value is NOT negative, just do a straight number conversion. IF the CALL PEEK or CALL LOAD value IS negative then subtract the value from Zero (negative number conversion) and look at the result. So lets convert the above result back into its Hex location.

Example - Convert CALL PEEK(-31776,A,B) into its Hex location.

```
             hex  0000  - 7C20   =  000083E0 ——— Result

Input        dec  00000 - 31776  = 0000033760
```

So CALL PEEK(-31776,A,B) is peeking into >83E0

NOTE: The Decimal input fields perform a range check as you input values. So, if you try to input a Decimal value greater than 65535 the screen will change to its Error Condition Colors.

---------- Explorer Options ----------

Cru Base 1100 0

Cru Base

Cru Base Switch

The Cru Base function of the Explorer allows you Read and Set the different Cru Bits in the 99/4A. Cru Bits are just a bit level method of I/O but they can be joined together to form bytes or words. In the console they are used for interrupt detection, keyboard and joystick input and the cassette controls. In the peripheral devices they can be used for anything the hardware and software would like. In many cases they are used as binary switches to turn on and off selected items. For example, in the disk controller card there are designated bits (switches) to select the proper drive, side and to strobe the motor on.

By setting the Cru Base field to a specified Cru address you can Read the Cru bit in the bit field. Also, after the specified Cru address has been set you can move the cursor to the bit field and input a 1 or a 0 to turn On or Off the specified bit (depending on the particular hardware).

The Explorer ALWAYS reads and redisplays the specified bit, however, some Cru bits can be set but not read as being set. This is a property of the hardware that the Cru Base address is set on. For example, if you have a TI RS232 Card you can turn it on by writing a 1 to Cru Base 1300. But, the Explorer will read back a zero even though the card is on. Now set the Cru Base address to 130E and input a 1 into the bit field. This will turn on the RS232 Card light and the Explorer will read back a 1. To turn the card off simply input a zero in the Bit field for Cru Base 130E and Cru Base 1300.

The value that is placed in the Cru Base address field is the same as the one you would place in Workspace Register 12 (R12) for a TB 0, SBO 0 or SBZ 0 instruction. To obtain this address you multiply the Cru Bit number that you want to Read or Set by 2 and add the Cru Base address to it.

Examples: Read Cru Bit 11 in the RS232 Card.

2 * 11 = 22   22 = >16   >1300 + >16 = Cru Base 1316

What Cru Bit number turns on the RS232 light?

>130E - 1300 = >0E   >0E = 14   14 / 2 = Bit number 7

In many locations in the 4A operating system you will find instructions like - TB 0   JNE >xxxx   These instructions are testing Cru bit number 0 of the Cru Base that is in R12 of the Workspace. The test result is reflected in the E (Equal) bit of the Cpu Status register. If the E bit is a 1 after the TB 0 instruction is executed the Cru bit was set (1) if it is a 0 the Cru bit was not set (0). You can fool the Application Program by changing the E bit immediately after the TB instruction is executed.

Cru Base Continued

Many of the peripheral cards and devices contain a Device Service Routine (DSR). This is a fancy name for the 9900 Assembly Language Object code that is burned into a Rom chip on the card or device. When the card or device is turned on this DSR code is paged into Cpu memory at >4000 through >5FFF so the 9900 micro can access and execute it.

With the Explorer it is a simple matter to find out where all of the DSRs are located in your system. TI set up the Cru Bases and peripheral cards in such a way that each card resides on a >100 boundary starting at Cru Base >1000. So the first DSR can reside at Cru Base >1000, the next at >1100 followed by >1200 etc.

To find the different DSRs in your system follow these steps.

1. On the Main Screen set the memory pointer to c4000s with ASCII display (Basic Bias Off).
2. Press FCTN 7 - move the cursor down to the Cru Base field and input 1000.
3. Move the cursor to the Bit field and input a 1.
4. Press FCTN 7 and look in the memory window. If its all dots there's no DSR at this Cru Base. If there is something there use the FCTN 4 Page Up and FCTN 6 Page Down to look around.
5. Someplace near the beginning (>4016) you should find the different DSR Link Names in the code (ie: DSK1, RS232, PIO, TP, etc.) or the CALLs that the card may contain (CALL FILES etc). This and the Cru Base map on the next page will help you identify which card's DSR you are looking at.
6. When you are finished looking at this DSR press FCTN 7 and input a zero in the Bit field to turn the card Off.
7. Move the cursor to the Cru Base field and input the next Cru Base address (add >100 to the previous Cru Base address) and continue with number 3 in these steps.

NOTE: The DSRs are ALWAYS turned on by writing a 1 to the lowest Cru Base address for that device (ie: 1000, 1100, 1200, 1300 etc.). The other Cru Base addresses in the range for the card are the 128 Cru bits assigned to each peripheral space (ie: 1102, 1104, 1106 etc.). But most of the peripheral devices do not use all 128 bits. They usually only use 6 - 24 bits. Also, the selected card's light may or may not come on when you turn on the DSR. It depends on the hardware design.

CRU BASE ASSIGNMENT MAP

Cru Base Continued

Cru Base Assignments

Cru Base	TI Assignment	Your System
0000 - 03FE	Internal Console Use (see Appendix)	
0400 - 0FFE	unassigned (not scanned by DSR Link)	
1000 - 10FE	unassigned - Production Tester	
1100 - 11FE	Floppy Disk Controller	
1200 - 12FE	Internal Modem	
1300 - 13FE	RS232 1 & 2 & PIO 1	
1400 - 14FE	unassigned	
1500 - 15FE	RS232 3 & 4 & PIO 2	
1600 - 16FE	unassigned	
1700 - 17FE	Unreleased HEX-BUS Adapter	
1800 - 18FE	TI Thermal Printer	
1900 - 19FE	EPROM Programmer	
1A00 - 1AFE	unassigned	
1B00 - 1BFE	Unreleased TI Debugger Board	
1C00 - 1CFE	Video Controller	
1D00 - 1DFE	IEEE 488 Controller Card	
1E00 - 1EFE	unassigned	
1F00 - 1FFE	P-Code Card	

The Cru Base addresses go up to >1FFE, Cru Base 2000 wraps around so you are actually looking at 0000. To test this, input a Cru Base of 3100 (1100+2000) and write a 1 to the Bit field and your disk controller light will come on. Don't forget to turn it back Off.

NOTE: To enable RS232 3 & 4 and PIO 2 requires a special modification to your RS232 Card. The DSR Link names are in the Card but the hardware must be modified to place it at Cru Base 1500 to access them. This modification allows you to have 2 RS232 Cards in your system for a total of 4 RS232 ports and 2 PIO ports.

Load Characters	h	s step speed
Execute Key Scan	h	h high speed

Options

Load Characters

The Load Characters Option of the Explorer will place character set loading in one of two modes. The H mode loads the character sets at full speed. This is very handy for Basic and Extended Basic since they Reload the character set after each completed command in the Command/Edit mode or at the completion of a running program. When the mode is set to S the Explorer will track each byte of the character set as it is moved out to Vdp Ram. After you have watched the lengthy process of loading a character set a few times (S mode), we are sure that you will appreciate the default H mode.

NOTE: The H mode has no effect when an Application Program, such as the Power Up routine or the TE II Module, DOES NOT use one of the GPL subroutines for loading the character sets.

Execute Key Scan

The Execute Key Scan Option of the Explorer allows you to speed up the key scan process by turning off the debounce delay loop. When this Option is set to the H mode the Explorer will automatically zero out the key scan's debounce delay loop counter and continue on with the key scan routine. If this Option is set to the S mode the Explorer will NOT zero out the debounce delay loop so inputting key strokes becomes a very slow operation.

If you would like to watch the complete key scan routine simply place an S in this field. However, after a few times through the key scan we're sure you will prefer the H mode for most Application Program execution.

Since the Explorer is operating your computer and Application Program in interpretive mode the inputting of key strokes is slower than normal. To properly input key strokes with the Application Program screen displayed just hold down the desired key until the cursor disappears. When the cursor reappears you can input the next key stroke. If you are inputting two or more consecutive key strokes of the same character wait for a second after the cursor reappears before pressing the key again. This prevents the application programs auto repeat delay counter from activating. When it does active it takes a long time before the cursor disappears. If it does activate just let up on the key, wait a second and then press it down again.

Colors	text/scrn	Counter
Status Screen	F4	
Break Point	1A	d00000000000
Error Condition	F6	

Color Options & Counter

Colors

The Colors Option of the Explorer allows you to set your own Text and Screen colors for the Explorer's Status screens (all screens). You can also set your own Break Point colors and Error Condition colors. The Break Point colors are the colors that the Explorer changes the screen to whenever a Programmable Break Point is encountered. The Error Condition colors are the colors that the Explorer changes the screen to whenever you try to input an invalid value or press an invalid key.

These colors are set in a 2 nibble field. The first nibble is the Text color and the second nibble is the Screen color. The color values are the same as the ones used in Assembly Language (see table below) and match the values that would be written to Vdp register 7 for Text Mode. NOTE: If you type in the same value for the Status Screen's Text and Screen colors you will not be able to see the Text any more. However, the Explorer will not leave the field until you press Enter or an Arrow key so you can just input a different value for the screen color and your Text will reappear, if its not transparent (0)

Color Values

0 = Transparent	8 = Medium Red
1 = Black	9 = Light Red
2 = Medium Green	A = Dark Yellow
3 = Light Green	B = Light Yellow
4 = Dark Blue	C = Dark Green
5 = Light Blue	D = Magenta
6 = Dark Red	E = Gray
7 = Cyan	F = White

Save Options - CTRL 9

After you have set the the H S Options and Colors you can save these settings to the Explorer Disk. The Explorer will then use these settings as the default settings whenever it is loaded. To Save your settings, first make sure the Options Screen is displayed. Then remove the write protect tab from the Explorer diskette. Next place the Explorer disk BACK IN DRIVE 1 and press CTRL 9. And finally, put the write protect tab back on the Explorer disk. Your settings will not be saved if the Explorer disk is not in drive 1 or the write protect tab is on the disk.

Counter

This display area is the decimal conversion of the Hexadecimal Counter on the Main Screen. You can not edit this field, it was placed here so that you didn't have to convert the Hex counter yourself.

EXPLORER'S REGISTERS SCREEN

```
------------ Status Registers ---------

   hgh grt cnd car ovf
Gs  0   0   0   0   0   0   0   0   00


   int 5rw cnc fifth sprite number
Vs  1   1   0   0   0   0   0   0   C0

------------ VDP  Registers ---------

                           bit ext
v0  0   0   0   0   0   0   0   0   00

   16k scn int txt mlt     sze mag
v1  1   1   1   0   0   0   0   0   E0

v2 0000 0000   00   screen image  0000
v3 0000 1110   0E   color  table  0380
v4 0000 0001   01   char pattern  0800
v5 0000 0110   06   spri attribu  0300
v6 0000 0000   00   spri pattern  0000
v7 1111 0101   F5   txt/scrn clr
```

On the Explorer's Registers Screen you will find the hex value and binary break down of the Gpl Status byte, the Vdp Status Register and Vdp Registers 0 and 1. This screen also contains the hex value, binary value and the actual table addresses for Vdp registers 2 through 6. For completeness Vdp Register 7 was also included here.

Not all of the bits in the Gpl Status byte and Vdp Registers 0 and 1 are active so only the active bits are labeled. As you move the cursor around on this screen the Explorer will automatically skip over the inactive bits. Also, since, the Vdp Status is a Read Only Register you can not edit this register it is here for the binary break down.

Even though all of the bits are not active in Vdp Registers 2 through 6 you can still edit the inactive bits but they will not effect the Table address. This was allowed since some Application Programs write values that turn on the inactive bits even though they are ignored by the TMS 9918A Vdp Processor.

```
----------- Status Registers ---------

   hgh grt cnd car ovf
Gs  0   0   0   0   0   0   0   0   00

   --- --- --- ----- ------ ------
--  -   -   -   -   -   -   -   -   --
```

Gpl Status

The Gs display breaks down the Gpl Status byte into its binary representation. This byte is not an actual hardware register, like the Cpu Status and Vdp Status. It is instead controlled and used by the Gpl interpreter software in console Rom. The actual location of this byte is at >837C in Scratch Pad Ram. If you change the bits or the hexadecimal value displayed here, the Explorer will automatically change the value at >837C and the value on the Explorer's Main screen and visa versa.

The following bits are used by the Gpl interpreter:

Hgh - High
When this bit is set it indicates to the Gpl interpreter that a Gpl Logical Greater Than (unsigned numbers) condition exists.

Grt - Greater Than
When this bit is set it indicates to the Gpl interpreter that a Gpl Arithmetic Greater Than (signed numbers) condition exists.

Cnd - Condition bit
This bit is used by the Gpl interpreter for a variety of items. Its main use is to indicate a True (1) or False (0) result of a test. These tests are Gpl instructions that move one of the other bits in this byte to the CND bit. It is this bit that the BRANCH ON SET (>60) and BRANCH ON RESET (>40) instructions test before branching. Also, the Key Scan routine sets this bit if a key press is detected.

Car - Carry
When this bit is set it indicates to the Gpl interpreter that the most significant bit of the word or byte being operated upon has been carried out of the word or byte into this bit.

Ovf - Overflow
This bit gets set when a Gpl operation results in a too large or too small condition for 2's compliment numbers.

---------- Status Registers ---------

```
   --- --- --- --- ---
--  -   -   -   -   -   -   -   -   --

   int 5rw cnc fifth sprite number
Vs  1   1   0   0   0   0   0   0   C0
```

Vdp Status

The Vs display is the binary break down of the TMS 9918A Vdp Processor's Status Register. The value that is shown here represents the value contained in that register when the the Application Program screen was LAST displayed. Since this register is a hardware Read Only register you can not edit the bits or hexadecimal value. It is here to display the binary break down which includes the following bits:

int - Vertical Retrace Interrupt
When this bit is set it indicates that the TMS 9918A has started the vertical retrace period. Vertical retrace is when the raster scan reaches the end of the active display area and then moves back to the top of the screen. It is at this time that the Vdp chip generates the Vdp interrupt for the 9901. Vertical retrace happens 60 times per second so we have 60 Vdp interrupts per second on a 60 Hz system (50 per second on a 50 Hz system).

5rw - 5 or more sprites on a row
This bit is set by the Vdp chip whenever there are 5 or more sprites on a row and the INT bit is set to 0 (not checked during vertical retrace). Since sprites are not allowed in Text mode this bit has no meaning when Text mode is active.

cnc - Coincidence
This bit is set by the Vdp chip whenever two or more sprites have at least one overlapping pixel. The Vdp chip checks for overlapping pixels as it generates the pixels on the screen, so this check occurs 60 times a second (60Hz). (Note: No sprites in Text mode.)

fifth sprite number
Whenever the 5rw bit is set, and you are not in Text mode, these 5 bits contain the number of the fifth sprite on the row. If 5rw is not set or if you are in Text mode these bits have no meaning and usually contain garbage.

```
------------ VDP  Registers ----------

                                   bit ext
v0  0    0    0    0    0    0    0    0    00

   16k scn int txt mlt      sze mag
v1  1    1    1    0    0    0    0    0    E0

v2 0000 0000   00   screen image  0000
v3 0000 1110   0E   color  table  0380
v4 0000 0001   01   char pattern  0800
v5 0000 0110   06   spri attribu  0300
v6 0000 0000   00   spri pattern  0000
v7 1111 0101   F5   txt/scrn clr
```

Vdp Write Only Registers

The next 8 Vdp Registers are the Vdp Write Only Registers. This means that your software can write to these registers but it can not read them. The first two, V0 and V1, control the various Vdp modes. The next five, V2 through V6, control the various table locations in Vdp Ram and the last one, V7, controls the Text Mode text color and All modes screen color.

v0 -

Bit - Bit Map
This binary switch enables and disables Bit Map Mode (1 or 0) when TXT and MLT in V1 are Off.

Ext - External Vdp
This binary switch enables and disables the External Vdp chip synchronization option of the TMS 9918A. You can enable this bit but since there isn't an External Vdp chip hooked up to synchronize with your Application Program's screen will go out of sync. Since you are in control you can play with this bit without hurting the Application Program.

```
     16k scn int txt mlt     sze mag
v1    1   1   1   0   0   0   0   0   E0
```

Vdp Write Only Registers Continued

v1 -

16K - 16K Vdp Ram
When this bit is set the TMS 9918A Vdp Processor handles the Dynamic Ram refresh for 4108 (8K) or 4116 (16K) DRam chips. When this bit is zero the TMS 9918A refreshes the 4027 (4K) DRam chips. The 99/4A contains 4116 DRams, the 4108 and 4027 DRams are not used in our consoles. IMPORTANT NOTE: If this bit is zero and the SCN (Screen Enable) bit is also zero the 4116 Vdp Dram chips will not be refreshed properly. This causes the values in many areas of Vdp memory, including the screen image, to decay to zero when the Application Program screen is toggled in (CTRL 3).

scn - Screen Enable/Disable
When this binary switch is zero (disable) the Application Program screen will be blank and only the screen color will show. Many of the Application Programs blank the screen image while they are building their screens. To watch an Application Program build its screen just turn on this bit.

int - Vdp Interrupt Enable/Disable
When this binary switch is On (1) the Vdp chip will generate a Vdp Interrupt signal for the 9901 whenever the raster scan is at the end of the active display area (60 times a second for 60 Hz).

txt - Text Mode
Turning this binary switch On places the Vdp Processor in Text Mode (40 columns) when the BIT in V0 and MLT in V1 are zero.

mlt - Multi-Color Mode
Turning this binary switch On places the Vdp Processor in Multi-Color Mode when the BIT in V0 and TXT in V1 are zero.

sze - Sprite Size
This switch selects the Sprite Size for the Vdp chip. When it is On (1) all sprites are made up of 4 characters (16x16 pixels). When it is Off all sprites are made up of 1 character (8x8 pixels).

mag - Sprite Magnification
This switch selects the Sprite Magnification for the Vdp chip. When it is On the sprite pixels are magnified 2x. When it is Off the sprite pixels are normal size.

v2	0000 0000	00	screen image	0000
v3	0000 1110	0E	color table	0380
v4	0000 0001	01	char pattern	0800
v5	0000 0110	06	spri attribu	0300
v6	0000 0000	00	spri pattern	0000
v7	1111 0101	F5	txt/scrn clr	

Vdp Write Only Registers Continued

This section of the Explorer's Registers Screen contains information about Vdp registers v2 through v7. The Explorer automatically calculates the start address for each of the different tables that are controlled by v2 through v6. If you change the binary or hex values for these registers the address for that table in the Application Program will also change.

v2 - Screen Image Table

The least significant 4 bits (---- 0000) in this register control the start location of the Screen Image Table. This is the area of Vdp Ram that holds the characters you see on the screen. This start address is equal to the value of the least significant 4 bits times >0400.

v3 - Color Table

All of the bits in this register control the start location of the Color Table. The Color Table is the area of Vdp Ram that contains the foreground and background colors for each of the active character set groups. This start address is equal to the value in this register times >0040. Note: In Bit Map mode this register controls the ENDING location of the Color Table.

v4 - Character Pattern Table

The least significant 3 bits (---- -000) in this register control the starting location of the Character Pattern Table. This area of Vdp Ram holds the character definitions (CALL CHAR) for each of the active characters. The start address is equal to the value of these 3 bits times >0800. Note: Basic and Extended Basic set this table at >0000 - but the actual character definitions start at >03F0. This is really the the start of the definition of character number 126 not 30 - so Basic and Extended Basic Add 96 (>60) to the value of each character placed on the screen to compensate for this offset in the table. This is how the Basic Bias came to be and it was done by TI to conserve space in Vdp Ram. Note: In Bit Map mode this register controls the ENDING location of the Character Pattern Table.

v5 - Sprite Attribute Table

The least significant 7 bits (-000 0000) in this register control the starting location of the Sprite Attribute Table. This Table holds the Character number, Color, Dot Row and Dot Column position of each active sprite. The start address is equal to the value of the least significant 7 bits times >0080

```
v2 ---- ----   --   ------ -----  ----
v3 ---- ----   --   -----  -----  ----
v4 ---- ----   --   ---- -------  ----
v5 ---- ----   --   ---- -------  ----
v6 0000 0000   00   spri pattern  0000
v7 1111 0101   F5   txt/scrn clr
```

Vdp Write Only Registers Continued

v6 - Sprite Pattern Table

The least significant 3 bits (---- -000) in this register control the starting location of the Sprite Pattern Table. Note: In Bit Map mode this register controls the ENDING location of the Sprite Pattern Table. This table holds the character definitions for the characters that make up the sprites. In Basic and Extended Basic this table also starts at zero. So, this table and the Character Pattern Tables are identical. What this means is that a sprite that is made up of character number 65 is actually made up of an A. So if you redefine the A (CALL CHAR(65,...) the sprite will also change. In Assembly or Forth you can place this table in a different location so that the sprite definitions and the character definitions can be independent of each other. Since this table overlaps the Character Pattern Table it is also subject to the Basic Bias. So if you execute a CALL SPRITE(#1,65...) you will find >A1 (161) in the Sprite Attribute Table for sprite #1 not >41 (65).

In this Table and the Character Pattern Table there are 8 bytes per character. So to find the start address for a given character number just multiply the character number times 8 and add the tables starting address to it. If you are in the Basic or Extended Basic environment don't forget to add the Basic Bias (>60 or 96) to the character value before you multiply it times 8.

```
Examples: Basic or Extended Basic  "A"
        65 + 96 = 161       161 * 8 = 1288
        1288 = >0508        >0508 + >0000 = >0508 char pattern for "A"

        Editor Assembler or Mini Mem "A"
                            65 * 8 = 520
        520  = >0208        >0208 + >0800 = >0A08 char pattern for "A"
```

v7 - Text and Screen Color

This register sets the color of the text when Text Mode is active and the color of the screen in ALL Modes. In this register the most significant nibble (0000 ---- or >0-) sets the text color. The least significant nibble (---- 0000 or >-0) sets the screen color. When you execute CALL SCREEN(8) the Basics subtract one from 8 and write a 7 for the screen color. For some reason TI wanted the colors to start at 1 instead of 0 in the Basics so ALL the color codes are offset by -1 before they are written to the Color Table or this register.

THE INTERRUPT ROUTINE

As we mentioned earlier the Explorer WILL execute the Interrupt Routine along with the normal program flow if interrupts enabled. The Explorer executes the interrupt routine whenever the Interrupt Mask does not equal zero so this is a simulated interrupt and not an actual Hardware interrupt. The interrupt routine in the 99/4A performs the following functions (in this order):

First a LIMI 0 is executed to disable any other interrupts - since the interrupts perform a context switch the old LIMI value along with the status is in R15 of the Interrupt Workspace (83C0). Note: The Explorer automatically sets the Interrupt mask to zero when RTWP is executed.

Next the Workspace is changed to the GPL Workspace (83E0) and R12 is cleared.

The Cassette interrupt timer flag in R14 is checked to see if it is On. If it is the interrupt routine hops down to the Cassette Timing routine at >1404 to continue which does NOT service the rest of the items.

If the Cassette interrupt flag was not set Bit 2 of the 9901 is checked for a Vdp interrupt. If it was a Vdp interrupt instead of an External interrupt (generated by a peripheral device) the routine continues at VDP INTERRUPT.

EXTERNAL INTERRUPT - If it was an external interrupt (peripheral) the routine turns on the cards one at a time starting at Cru Base 1000 to check for interrupt routines. If the card contains an interrupt routine (like the RS232 card) it is executed. Then the card is turned off and the next card is checked until the Cru Base = 2000 (end of peripheral devices). Then it hops down to END and leaves the interrupt routine. So, an External interrupt does NOT service the rest of the items.

VDP INTERRUPT - First bit 2 of the 9901 is reset (turned off), then 83C2 is moved into R1 and the most significant bit is checked. If its On then the routine hops down to VDP STATUS and skips Auto-Motion, Auto-Sound and the QUIT key.

AUTO-SPRITE MOTION - The Auto-Sprite Motion bit in R1 is checked and if it is On this routine is skipped. Otherwise the interrupt routine moves 837A (highest sprite number in auto motion) into R12. If it is zero the rest of the routine is skipped otherwise the routine moves the sprites to their new location according to the Sprite Motion (0780) and Sprite Attribute Tables in Vdp Ram.

AUTO-SOUND PROCESSING - The Auto-Sound Processing bit in R1 is checked and if it is On this routine is skipped. Otherwise the interrupt routine checks the Most significant byte in 83CE. If its zero the rest of the routine is skipped if its not the location of the sound table is checked (Vdp Ram or Grom) and the next byte of the table is moved to the sound chip and 83CE is decremented.

QUIT KEY - The QUIT Key bit in R1 is checked and if it is On this routine is skipped. Otherwise the FCTN and = Keys are scanned. If they are being pressed down it performs a software reset (BLWP @>0000 - Power Up routine)

VDP STATUS - If the most significant bit in 83C2 was set the interrupt routine would have hopped down to here to continue execution. At this point the Vdp Status byte is copied to 837B

The Workspace pointer is set to 83C0 and the Screen Time Out Counter is then incremented by TWO. Next it is checked, if it is NOT zero the rest of this routine is skipped. Otherwise this routine, which is also pointed to by the Level 2 interrupt vector, is executed. This routine uses the copy of Vdp Register 1 which is stored at 83D4 (R10 of the Interrupt Workspace) to blank the screen by masking out the SCN bit in Vdp Register 1.

The Workspace pointer is set back to 83E0 and the Interrupt Timer at 8379 is incremented by the value in the most significant byte of R14.

The User Interrupt Vector (ISR Hook in 83C4) is moved into R12 and if its not zero the User Interrupt routine is BRANCH and LINKed to. (R11 of 83E0 contains the return address). After it is completed it sets the workspace back to 83E0 and returns with a RT (B *R11) instruction.

END - R8 of the Gpl Workspace (83E0) is cleared, the workspace pointer is set to 83C0 and a RTWP is executed.

EXPLORATIONS

In this section of the manual we will take you through a few Explorations on the operation of the 99/4A. Keep in mind that since the Explorer is a machine language interpreter the Application Programs run slower. The Explorer operates the computer at about 1/300 its original speed, with the Application Program's screen displayed and interrupts disabled (CTRL 4). With interrupts enabled the speed can decrease to about 1/1000 the original speed. With this in mind best case tells us that something that normally takes 1 second will now require approx 300 seconds to complete. This is very evident when you are executing the Basic languages because they are now going through three levels of interpretation. The Explorer interprets the machine language which is the Gpl interpreter interpreting the Gpl object code which contains the Basic interpreter which is interpreting the Basic commands and/or program. And this is why we recommend that you use the Explorer on the Basics with direct CALLs and small programs. The heart of the Explorer, the machine language interpreter, has been optimized for speed and efficiency but whenever something is executed interpretatively it slows down. However, we felt that the wealth of information the Explorer returns because of its interpreter was worth it. So lets get started, but please follow the Explorations in order.

First we will look up a few values in your computers memory. TI has released a number of different version of the 4A so we want to be sure that you use the right values. Most of the consoles have the same Rom but the Grom has under gone some minor changes. Only the real early 4A consoles have a slightly different Rom.

1. Load the Explorer into the Extended Basic environment.

2. Set the WS field under Cpu to 0000. With the WS set here the Workspace registers contain the data that is in Cpu Rom memory at >0000. We had you use the WS instead of the Memory Window because it is easier to see the Vectors in word form.

3. The registers currently contain the following information:

R0 = The Workspace Pointer (WS) for the Power Up Routine
R1 = The Program Counter (PC) for the Power Up Routine
R2 = The Workspace Pointer (WS) for the Level 1 Interrupt Routine
R3 = The Program Counter (PC) for the Level 1 Interrupt Routine

R4 = The WS for a routine that blanks the screen
R5 = The PC for the above routine
R6 = Data - Console clock speed and "AA" for checking validation bytes
R7 = The opcode for the Branch instruction (start of Assembly key scan)

R8 = The Cpu BP address to use for halting the Explorer on a Key Scan
R9 = Data - zero and eight

For the first Exploration just remember the value in R8 (02B2). A little latter on we will use the values in R0 & R1.

KEY SCAN AND THE EXPLORER

You already have one value that we need but we will need to SEARCH for one more. The value that was in R8 on the previous page is a good value to use for the Cpu Break Point for any Key Scan. You see, there is a slight difference between the entry point for the Key Scan that an Assembly Language program uses and the entry that the Gpl Interpreter uses. But, they both pass through the R8 address. Note: Most consoles contain >02B2 as the address in R8 when the WS is set at >0000.

Now lets find the second value that is handy to know for Key Scan operations. This value will be the PC address when a key press has been detected and the proper Cru Bits from the keyboard have been converted into their key code Hex value. But the Hex value has NOT been moved out to >8375 in Scratch Pad Ram yet, it is still in R0 of the Workspace. By setting your Break Point here you can change the value in R0 and fool the Application Program into thinking that a different key was pressed. This is handy if you want to input CTRL 2 through CTRL 5, which are used by the Explorer for special controls.

1. Set your Memory Pointer to c----s. Any address will do since we are going to activate the Search Function.

2. Press FCTN 5 to activate the Search Function and input 0000 as the ST address and 1FFF as the FN address.

3. Make sure your Memory Window display is in Hex mode (FCTN =)

4. Place your cursor in the Search String input area and type in 70 20.

5. Press the left arrow key once to place the cursor on top of the 0 in 20 and then press Enter.

6. When the Explorer finds this value REMEMBER the Memory Pointer address. For most consoles it should be >043E.

7. Press FCTN 5 to turn off the Search Function and Press FCTN 7 to bring up the Number Converter.

8. Input >043E (or your Memory Pointers address) into the first Hex input field and >0006 into the second Hex input field.

9. If the Operation Indicator is not a + then press FCTN 9 to place the cursor on the + and press Enter.

10. The value after the = in the Hex result area is the Cpu Break Point address we were looking for. >0444 for most consoles.
Now write these two values (addresses) down for future reference.

02B2	0444
Key Scan BP (02B2)	Key Press Detected (0444)

Now that you have these two values (addresses) you can easily use them as Break Points in your Explorations. But you should be aware that the Application Program is not always looking for a key press when the >02B2 is reached. Sometimes the Application Program is just resetting the keyboard (9901) to a known state. You can count on the second address (>0444) as always indicating that a key press has been accepted. If you want to use CTRL 2 - CTRL 5 in executing your Application Program simply set the Cpu Break Point to the second address. When the Break Point is reached you can replace the key code in the most significant byte of R0 with the appropriate key code from the following table:

CTRL 2 = >B2   CTRL 3 = >B3   CTRL 4 = >B4   CTRL 5 = >B5

So now lets see how those addresses work.

1. Set your WS back to 83E0, make sure your PC is at 006A, set the Cpu BP at 02B2 (or your value) and press CTRL 2 and CTRL 3.

2. The Explorer will now start to return to the Command Mode of Extended Basic. It will reach the Break Point before the screen scrolls but this is just Extended Basic resetting the keyboard. Just press any key to release the Break Point condition and press CTRL 2 and then CTRL 3 to start it back up.

3. When the Break Point is reached again Extended Basic is in Command mode and is looking for a key press. Press any key to release the break point condition.

4. Set your Memory Pointer to g----d with FCTN 1 so you can watch the key scan grab the key code out of of Grom 0.

5. Set the Cpu BP at >0444 (or your value), press and hold down CTRL 2 and then press and hold the "L" key.

6. Release the CTRL and 2 keys at the same time but hold down the "L" key until the Break Point is reached.

7. The Explorer will then execute the key scan and Break when the key is detected and decoded, watch R4 when the key is detected. The most significant byte in R0 is the key code for "L" and should be >4C. Change it to >4D, the "M".

8. Now press CTRL 3 and then press CTRL 2 to start up the Explorer. Extended Basic will now place an "M" on the screen instead of the "L".

NOTE: If you use the CTRL 1 - Single Execution - through the key scan the CTRL key may be detected, so if you were holding down the "L" key it would be decoded as CTRL L instead. The code that actually reads the keyboard cru bits is STCR R4,8   INV R4 (SEARCH for opcodes 36 04 05 44 in console Rom for this Break Point, it should be at >0346.)

THE POWER UP ROUTINE

For this Exploration it doesn't matter what environment the Explorer is loaded into. Also, if the Explorer is already loaded, it doesn't matter where its at in the Application Program because we are going to restart the computer.

The Power Up routine can be executed at anytime by simply following steps 1 & 2 and starting up the Explorer. What we are going to do though is set a few Break Points along the way. Remember the R0 and R1 values we obtained when the WS was at 0000, well its now time to use them.

1. Set the WS at 83E0 (R0 value) if it is not already there.

2. Set the PC at 0024 (R1 value).

3. Set the Grom BP at 00EB

4. Move the cursor down to the R0 field and press and hold the zero key to clear out the WS. (This is not necessary for proper execution of the Power Up routine but we wanted you to see the first few instructions work)

5. Set the Memory Window to g----d and set the Instruction Counter to zero.

6. Set the disassembly to a 3 line display (FCTN 3) and make sure the interrupts are disabled (CTRL 4).

7. Now press CTRL 1 a few times and watch the Workspace as the instructions start to set it up.

8. When the PC reaches 0060 the Gpl interpreter is about to set the Grom address to the value in R6 (0020). These are the set Grom address instructions we talked about in the Grom Controls section of the manual. Press CTRL 1 and watch the Grom address and Memory Window as each byte of the address is passed to the Grom Write Address port (9800 + 402 = 9C02).

9. After the address is set it will clear the Gpl Status CND bit (PC=006A) and then Set the Interrupt Mask to 2 and then to 0. If you had interrupts enabled (CTRL 4) the Explorer would go out and execute the interrupt routine right after the IM was set to 2.

10. The Grom address of 0020 is the start of the Gpl Power Up routine so now lets turn it on, CTRL 2 and then CTRL 3 and let it reach our Break Point.

11. When the Break Point is reached (approx 9 seconds) just press any key to release the Break Point condition. This Break Point was set to allow you to turn the screen on. Four times previous to this break point the power up routine reset Vdp register 1 with the SCN bit off but now it will leave it where we set it.

12. You can turn the screen on for Graphics mode (32 column) by either writing E0 to v1 or by pressing FCTN 8 and moving the cursor down to the SCN bit in v1 and turning it on. Then press FCTN 8 again to get back to the Main screen.

13. Set the Grom BP to 6000, set the Vdp BP to 4300 and start up the Explorer again (CTRL 2 CTRL 3). The Power Up routine will now start to clear out the first 4K of Vdp Ram. Our Vdp Break Point will be reached in approx 20 seconds.

14. When this break point is reached clear the break point condition by pressing any key and set the Vdp BP to 5000. Next Set your Memory Pointer to v0000s and the display to ASCII (FCTN =) with Basic Bias Off.

15. Press FCTN 9 to put your cursor in the Memory Window (Editor) at 0000 and type in "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ". Next press CTRL 3 to see the Application Program's screen. You may or may not be able to something on the screen at this time depending on what is in Vdp Ram where the tables are currently set at. We had you do this so you could watch the Power Up routine load the Title screen character set. The Power Up routine does not use the built in subroutines to do this so the Load Characters H S option has no effect.

16. The Power Up routine still has to clear the rest of the first 4K of Vdp Ram so it will be approx 98 seconds before the character set starts to load. The Vdp BP of 5000 will Break the Explorer before this happens so you can go take a break. The loop counter in R8, which should be at 0D01, counts down the number of bytes yet to be cleared. You can bypass this clearing operation by setting R8 to 0001 but the Color Bars on the title screen may contain garbage because their character definitions haven't been cleared out. (Don't reset R8 for this Exploration)

17. When the Vdp BP of 5000 is reached clear the Break point condition and set the Vdp BP to FFFF (turn it off). The Power Up routine still has a couple of items to do before it loads the character set. First it will load the color table and then it will scan the keyboard a few times to set it to a known state. This takes approx 5 seconds. Then it will load the character set. If you want to watch this happen on the Main Screen set the Memory Pointer to v----d in Hex display mode and the Explorer will dynamically track the bytes as they are written to Vdp Ram.

18. Press CTRL 2 CTRL 3 and watch the Title screen character set load. While it is loading you can press CTRL 3 to toggle between the Main Screen and the Application Program's screen. After the character set is loaded the Power Up routine will clear the screen and start to build the Title screen. The Grom BP of 6000 that we set earlier will Break the Explorer when the Title screen is fully built.

19. When the Explorer reaches the Grom BP clear the break point condition and set the Grom BP to FFFF also set the Cpu BP to 02B2 (or your key scan break point).

20. Set the Memory Pointer to v0128s in ASCII display mode. Press FCTN 9, type in your name and the press CTRL 3 to see YOUR new Title screen. Now lets play with the Vdp registers a little. Press CTRL 3 to bring up the Main screen again.

21. Press FCTN 8 to bring up the Registers Screen. Move the cursor down to the BIT switch in v0 and enter a 1 (turn it On). Now press FCTN 8 and then CTRL 3 to see the Title screen in Bit Map mode.

22. Press CTRL 3 to bring up the Main Screen, press FCTN 8 and turn Off the BIT switch. Move the cursor to the TXT switch in v1 and turn it on. Press FCTN 8 and then CTRL 3 to see the Title screen in Text Mode.

23. Press CTRL 3 and then FCTN 8, turn Off the TXT switch and turn ON the MLT switch. Press FCTN 8 and CTRL 3 to see the Title screen in Multi-Color mode.

24. Press CTRL 3 and then FCTN 8, Leave the MLT switch ON and move the cursor back to the BIT switch and turn it On. That's right Bit Map Multi-Color mode - the one that TI forgot to tell us about! Now press FCTN 8 and CTRL 3 to see the Title screen in the new mode. One other new mode is Bit Map Text mode - which is Bit Map mode with the screen trying to display 40 columns. We haven't actually used these new modes for anything but they might be fun to play around with to see what the results would be.

25. When you are done playing with the various modes set the BIT, TXT and MLT switches Off to set the Title screen back to its normal Graphics (32 column) mode. If you know the values to place in v0 and v1 you can also set these modes from the Explorer's Main screen by editing these fields.

26. Take a look at the Instruction counter and press FCTN 7 to see it in Decimal. Ours says 105,272 Machine Language Instructions have been executed so far! And that does not include and interrupt routine execution since we disabled it (CTRL 4). Press FCTN 7 again to bring up the Main Screen.

27. Press CTRL 2 to start up the Explorer again but leave it on the Main Screen. In a few seconds your Disk Controller light will come on. What has happened is the Power Up routine has started its peripheral card scan and it is looking for cards that contain Power Up Routines. The Disk Controller contains a Power Up routine that sets up Vdp Ram to a default of CALL FILES(3). IMPORTANT - If you are using a Myarc RAM DISK card the Explorer will lock up at this point because the card does a 32K Bank Switch when its internal Power Up routine starts.

28. The TI DSR and our DSR (corcomp card) clear out Vdp Ram from 37D8 to 3FFF and then place some values at 37D8 to signify that a Disk Controller is in the system and that space is reserved for 3 disk files. The Myarc Disk Controller DSR clears out its own DSR Ram at 5000 to 57FF and then writes a few bytes to Vdp Ram 37D8.

29. After this clearing is finished the TI and Myarc cards continue with a normal Power Up. Our DSR stays in control and builds the 9900 Disk Controller Title screen and this is where the Power Up routines start to differ. At this point the TI and Myarc cards release control back to the Gpl interpreter, our DSR stays in control until you select Basic or a module from the menu or press the space bar. This probably wasn't the smartest thing to do since it doesn't allow a Power Up routine in the module to be executed prior making a selection. Only a few modules have Power Up routines in them like the TEII and if you press the space bar twice to bring up the normal TI menu instead of the 9900 DC menu the power up routines are executed.

30. At this point the TI and Myarc cards have released control back to the Gpl interpreter to continue on with a little house keeping. Our DSR is still in control and, like the other cards, it will soon be looking for a key press.

31. When the Explorer reaches the Cpu Break Point of 02B2 that we set earlier clear the Break Point condition. If you have our DSR (corcomp card) set the Cpu BP to FFFF, start up the Explorer, CTRL 2 CTRL 3 and press the space bar. When the screen starts to clear press CTRL 2 CTRL 3 and set the Cpu BP back to the key scan BP, 02B2. Then start the Explorer back up CTRL 2 CTRL 3. When the Cpu BP is reached again all of the Power Up routines will be in the same place no matter which Disk Controller card you have.

32. Clear the Break Point condition, set the Cpu BP to FFFF and start up the Explorer, CTRL 2 CTRL 3. At this time the Power Up routine is in a loop waiting for a Key press before it continues on. Go ahead and press a key.

33. After the key is pressed the console Power Up routine will scan the module that is plugged into the Grom port for a module Power Up routine. If it has one (most don't) it is executed and if it returns, instead of taking control, the console Power Up routine will build the menu (with the SCN bit turned off again) and then turn on the screen and wait for you to make a selection.

34. For now select 1 - TI Basic by pressing and holding the 1 key for a second or so (the screen will start to clear). The Power Up routine is still in control and it will now clear the screen. Normally at this time there is a BEEP sound generated but we have the interrupts turned off so this will not happen. But, the BEEP routine places a value of 0100 in 83CE and it loops and waits for this to be changed to zero by the interrupt routine before it continues. Toggle to the Main Screen (CTRL 3) and look at R3 if it contains 83CE then your are in this loop. If it doesn't contain 83CE then you are not there yet but you will soon be.

35. When R3 contains 83CE you can get out of the loop by simply pressing CTRL 5 - this turns off the sound generator and zeros out 83CE. You could also stop the Explorer, set the Memory Pointer to c83CE- and zero it out yourself but, CTRL 5 is much simpler because you can leave the Explorer running. NOTE: IF you make a selection from the 9900 Disk Controller menu (our DSR), other that the Space Bar, it will not go through this loop so you do not have to worry about clearing out the sound indicator at 83CE.

36. The Power Up routine will now clear out Vdp Ram from 1000 to the disk buffer space at 37D7. Once again R8 is the counter so you can bypass this lengthy process by placing 0001 in it. Why 0001 and not 0000, because the instruction flow decrements R8 and then jumps if its not zero. If it decrements 0000 you end up with FFFF and that's not zero so it will really mess things up. This is a GOOD rule to remember any time you change loop counters - always set them to at least 0001.

37. Once Vdp Ram is cleared or the counter reaches 0000 the Power Up routine clears parts of Scratch Ram, sets up the color table and releases control to Basic. You'll notice the Grom address to be in the 2000 through 57FF range (Groms 1 & 2) when this occurs. So that ends the Power Up routine and begins the next Exploration of TI Basic.

EXECUTING A BASIC CALL

There are 3 different ways to get into the Basic environment. The first and easiest is to load the Explorer there through CALL LOAD("DSK1.EXP" or MEXP) with the Editor Assembler or Mini Memory modules plugged in. The second way is to go through the Power Up routine as we did in the previous Exploration. And the third way is to set the Explorer up to directly execute Basic. (see Direct Execution of Other Modules). If you have just completed the previous Exploration then you are are already part way into the Basic environment. If, on the other hand, you are about to start your Explorations again then load the Explorer into the Basic environment.

> Start up the Explorer, CTRL 2 CTRL 3, and let it run until the screen scrolls and the cursor reappears. You are now in Basic's Command/Edit mode and it is waiting for you to type something in. When you type remember that everything is being executed interpretatively so things happen slower.

Executing A Basic CALL

1. After the cursor has reappeared press CTRL 2 and CTRL 3 to stop the Explorer and bring up its Main Screen. Now lets take a look at the format for a BASIC Subprogram Header (X-Basic is different).

2. Set the Memory Pointer to g----s. Press FCTN 5 to activate the Search Function and input 2000 for ST and 57FF for FN. 2000 - 57FF is the Grom address range for the Basic Interpreter. Put your cursor in the Search String input field with the display in ASCII and Basic Bias Off. Type in COLOR, press the left arrow key once to put the cursor on top of the R, and press Enter.

3. When the cursor reappears and COLOR is found press FCTN 5 to exit the Search Function. Then press FCTN 9 to put the cursor in the Memory Window. With the cursor sitting in the home position (first byte) of the Memory Window press the left arrow key 5 times. This will place the beginning of the Subprogram Header in the home position of the Memory Window. Now press FCTN = to display it in Hex, and here's what you have:

first word	=	4D38	pointer to next subprogram header
second word	=	5713	entry point for this subprogram
next byte	=	05	length of this subprogram's name
following bytes	=	434F4C4F52	COLOR - this subprogram's name

We are most interested in the second word or the Entry Point for the subprogram. This will allow us to set a Grom BP to halt the Explorer when Basic first starts to execute the subprogram. Previous to Basic arriving at this location it performs a few overhead type items which we will discuss in a moment.

4. Now lets Search for SCREEN, the one we are going to execute. Make sure the Memory Pointer is set to Grom memory. Press FCTN 5 to activate the Search Function and reset ST back to 2000. Put your cursor in the Search String input field with the display in ASCII and Basic Bias Off. Type in SCREEN, move the cursor on top of the N, and press Enter.

5. When SCREEN is found press FCTN 5 to exit the Search Function. Then press FCTN 9 to put the cursor in the Memory Window. With the cursor sitting in the home position press the left arrow key 5 times and set the display to Hex, now here's what you have:

first word	=	0000	no more subprogram headers
second word	=	37BF	entry point for this subprogram
next byte	=	06	length of this subprogram's name
following bytes	=	53435245454E	SCREEN - this subprogram's name

6. Press FCTN 9 to get your cursor out of the Memory Window and move up to the Grom BP field and type in the Entry Point address (37BF) for SCREEN.

7. Set your Memory Pointer to v----d or press FCTN 1 to bring up that Memory Window. Then press CTRL 2 and CTRL 3 to start up the Explorer.

8. SLOWLY type in CALL SCREEN(7). By slowly we mean press down the letter key and hold it until the cursor disappears. When the cursor reappears press down the next letter. When you get to the second L in CALL wait for a second after the cursor reappears before you press the L key again. If you press it too soon Basic will try to go into Auto-Repeat mode but before it repeats the key it has a LARGE delay loop. If the cursor doesn't disappear within a second or so after you press the second L just let up on the L key for a bit to get out of this Auto Repeat delay loop and then press it down again.

9. You may be wondering why the cursor doesn't blink. This is because the interrupts are turned Off. The Interrupt routine controls the counter that a lot of Application Programs use to blink the cursor and with the interrupts Off this counter never changes.

10. After you have the complete CALL SCREEN(7) typed in press CTRL 2 CTRL 3 and set the Cpu Break Point to the Key Detected Break Point (0444) then press CTRL 2 and CTRL 3. Next press and hold Enter until the cursor disappears and the Explorer Breaks. Clear the Break Point condition and set the Cpu BP to the Key Scan Break Point (02B2) and zero out the Instruction Counter field. Press CTRL 2 CTRL 3 to start it up again. Now here's where that overhead starts that we talked about.

11. Basic will now go through its CRUNCH routine. This routine parses the line you just typed in a word at a time and places the word in the CRUNCH buffer at 0320 in Vdp Ram. So first it will place CALL in the buffer and then it will scan through the reserved word list in Grom until it finds it. It then takes the token code for CALL (9D) and places it in the CRUNCH buffer where the C is.

12. After it has crunched CALL it will set up the Crunch buffer with the codes for an unquoted string of x characters (C8 xx) and move the word SCREEN out there. As each character of SCREEN is moved the x char value is updated. Then it will place the token for "(" (B7) and then the tokens for another unquoted string of 1 character (C8 01) followed by the 7 and finally the ")" (B6). By toggling your screen (CTRL 3) back and forth you can look in the Dynamic Vdp Memory Window and watch this all happen. Also since you are in control you can stop the Explorer at any time and examine different areas of memory and then start it back up again. You may want to change your Memory Window to Dynamic Grom or Hex for awhile. Note: The characters on the Application Program screen have the Basic Bias added to them but the items in the Crunch buffer do not.

13. When it has completed its crunching Basic will scroll the screen up one line and then begin execution of the CALL routine. CALL will in turn scan the peripheral DSRs and then Grom for SCREEN. Once SCREEN is found and its start address (entry point) is obtained it will save the return address, set up the start address and the Explorer will Break.

14. At this point release the Break Condition. You can now single step through the SCREEN subprogram or just turn on the Explorer and let it go. The SCREEN subprogram is approx 1,685 machine language instructions long.

15. After the SCREEN subprogram is complete Basic will return and execute the following housekeeping routines. Reload the character sets - reset the screen color back to Cyan - reload the color table - reset Vdp registers 2, 3 & 4 and then reset the keyboard, the Explorer will Break at this point but just start it back up again.

16. After it has performed its housekeeping it will scroll the screen up one line and bring out the cursor. When the cursor reappears you are back where we started in Command/Edit mode and the Explorer will Break. Clear the Break Point condition and press FCTN 7. That's right! 71,265 Instructions have been executed to perform a CALL SCREEN (without a variable or interrupts) in Command mode.

    You can use this Exploration for any Basic CALL. After seeing this it really makes one appreciate what goes on behind the scenes!

EXECUTING AN EXTENDED BASIC CALL

The Explorer that operates in the Extended Basic environment is slightly different in two ways. First it has its own special loader for Extended Basic and second when it is loaded it resides in a different area of High memory Expansion. This version leaves 6K of Extended Basic's program space in High Memory free for your programs. It is for these two reasons that you can not use the Extended Basic version with the Editor Assembler, Mini Memory or other Load and Run type Loaders. Also, since Extended Basic tries to load a file off of DSK1 named LOAD we do not recommend that you select XB from the menu (Power Up routine) unless you want to go through this lengthy process.

First load the Explorer into the Extended Basic environment Command/Edit mode with CALL INIT :: CALL LOAD("DSK1.XBEXP"), or MXBEXP for the Myarc card. Start up the Explorer, CTRL 2 CTRL 3, and let it run until the cursor reappears.

Executing AN Extended Basic CALL

1. When the cursor reappears stop Explorer and bring up its Main Screen. Now lets take a look at the format for an EXTENDED BASIC subprogram header which is different than a Basic or any other modules subprogram headers. Set the Memory Window for Grom (g-----) and Search for SCREEN in the module space (ST=6000 FN=FFFF)

2. Press FCTN 5 to leave the Search function and put the cursor in the Memory Window (FCTN 9) and press the left arrow 3 times to bring in the start of an Extended Basic subprogram header. Here's what we have in Hex display for Version 110:

first word	=	A088	pointer to next subprogram header
next byte	=	06	length of this subprogram's name
following bytes	=	53435245454E	SCREEN - this subprogram's name
next word	=	AC66	entry point for this subprogram (version 100 = AC5D)

As you can see the entry point for an XB subprogram header FOLLOWS the name whereas in the other module's subprogram headers it precedes the name length byte. This was done to keep the Extended Basic CALLs from being executed in Basic which has a different Vdp memory map (no sprites).

3. Now that we have the Entry Point for CALL SCREEN set the Grom BP to this value (AC66 - V110 or AC5D - V100), set the Memory Window to v----d and start up the Explorer with the Application Program screen displayed, CTRL 2 CTRL 3.

8. Slowly type in CALL SCREEN(7). After you have the complete CALL typed in stop the Explorer and bring up the Main screen. Set the Cpu BP to the Key Detected Break Point (0444) and start the Explorer back up.

9. Press and hold the Enter key until the Break Point is reached. Then set the Cpu BP to the Key Scan address (02B2). Next zero out the Instruction Counter and start up the Explorer with the AP screen displayed.

10. Extended Basic will now parse and crunch the CALL statement. It will also move it out to the Edit Recall Buffer in Vdp Ram at >08C0. You can use CTRL 3 to toggle between the screens and watch this happen. The crunch buffer (>0820) characters DO NOT have the Basic Bias (>60) added to them, but the characters on the screen (>0000 - >02FF) and in the Edit Recall buffer (>08C0) do.

11. When it has completed its moving, parsing and crunching Extended Basic will scroll the screen up one line and begin execution of the CALL routine. CALL will first scan Extended Basic's Grom for SCREEN. When you are in Command/Edit mode it will scan the peripheral DSRs and the Grom Library if its not found in Extended Basic's Grom. Once SCREEN is found and its start address (entry point) is obtained it will set up the Grom start address and the Explorer will Break.

12. At this point you can single step through the SCREEN subprogram or just turn on the Explorer and let it go. The Extended Basic SCREEN subprogram is approx 1,126 machine language instructions long.

13. After the SCREEN subprogram is complete Extended Basic will Return and execute the following housekeeping routines along with resetting some pointers: Reload the character sets - Reset the screen color back to Cyan (v7) - Reload the color table - Zero out the sprite motion table - Clear out the sprite attribute table - Reset Vdp registers 1 through 6 and then reset the keyboard, the Explorer will Break at this point but just start it back up again.

14. After it has performed its housekeeping it will scroll the screen up one line and bring out the cursor. When the cursor reappears you are back where we started in Command/Edit mode and the Explorer will Break. Clear the Break Point condition and press FCTN 7. This time it was only 32,651 machine language instructions to complete the CALL SCREEN (without a variable or interrupts). How come only half as many as Basic when Extended Basic has more housekeeping to do? (Hint - watch how Extended Basic scrolls the screen)

EXECUTING OTHER ASSEMBLY LANGUAGE PROGRAMS

Through the Explorer you can run other Assembly Language programs. This gives you a very powerful and simple to use debugging tool. It can also be used as a learning tool since you can see the immediate results of each and every instruction as they are executed. So, if the instruction says SRL R8,8 or XOP *R0,2 you can see exactly what happens to the Registers, PC and WS right after you press CTRL 1.

To load another Assembly Language program with the Explorer just keep the following items in mind.

* It must be a Load and Run - Non-Auto-Start type program. The PROGRAM (Memory Image) type files Auto-Start so these will not work.
* It must be loaded BEFORE the Explorer and it must NOT reside in the Explorer's memory space.

There is a file on the Explorer disk named PREASSEM. This file sets up a couple of pointers for the Editor Assembler and Mini Memory Loaders. These pointers are used by the loaders to determine where a program should load and if it will fit. To use this file simply select LOAD AND RUN from the menu and type in DSK1.PREASSEM. After this file has loaded it will return back to the Load and Run prompt. It has now set up the pointers to reserve space for the Explorer.

Next load your Assembly program and then the Explorer. With PREASSEM your program will not be allowed to load in the Explorer's memory space and it will return a MEMORY FULL Error if it tries to. This means that your program is too large to load into High Memory Expansion along with the Explorer. However, your program may load if it contains AORG statements BUT it will not run properly if any portion of it resides in the Explorer's memory space. If you do get a MEMORY FULL Error you will need to reload PREASSEM before trying the next file. Whenever there is an Error during loading the module resets the pointers.

The Assembly file will not load with PREASSEM installed if it does not contain any AORGs and it is larger than 6K bytes. By placing a couple of AORGs in your source code and reassembling the file file you can load up to 6K in High Memory Expansion and up to 8K in Low Memory Expansion. If you have the Mini Memory module plugged in you could also use its 4K of Ram. This gives you a total of 18K of space for your Assembly file.

NOTE: You do not need to use PREASSEM if you know for sure that your program is 6K or less in size. Using this file is just a simple way of making sure that the Explorer doesn't write over your program. This does not apply to AORG type programs since they do not use these pointers.

So Now lets load an Assembly Language program and run it through the Explorer.

For this example we will use the DEBUG program that comes with the Editor Assembler. This file is less than 6K in size so you do not need to use the PREASSEM file first. NOTE: If you are using the Mini Memory module you should REINITIALIZE it before you start.

1. Select LOAD AND RUN from the Editor/Assembler or Mini Memory menu and load DEBUG.

2. When the cursor comes back load DSK1.EXP, or MEXP for the Myarc card.

3. Press any key to get past the Explorer's Title screen and bring up the Main screen

   At this point you can start executing your Assembly program with one of two different methods.' The first method is the same as the one you normally use for all Non-Auto-Start programs. The second method simulates an Auto-Start program. Before you start this example write down the Grom Address and set the Cpu BP to BOBE (the start address for DEBUG)

Non-Auto-Start Method

1. Start up the Explorer with CTRL 2 CTRL 3. At this point the Application Program Screen is the LOAD AND RUN screen of the module.

2. After the Application Program has clear off the DSK1.EXP file name and the cursor has returned just hold down the ENTER key until the cursor disappears to bring up the PROGRAM NAME prompt.

3. When the PROGRAM NAME prompt appears (in approx 27 seconds) slowly type in YOUR Program's start name (DEBUG) and press Enter.

4. At this time the Editor Assembler module will do a few things before it starts to execute your program. It checks the name length - Moves the name from the screen into >834A - Loads the color table with >13 - Clears the screen - Sets Vdp Register 7 to >F3 - Scans the DEF Table for the name - Saves the start address at >2022 - Loads the Workspace pointer at >20BA - puts the start address in R0 and BRANCHES *R0 to start the program.

5. At this point the Explorer will Break. Clear the Break Point condition and write down the new Grom Address. We will talk about the Grom Addresses in a minute. Now you can start the Explorer back up or use CTRL 1 to see how the DEBUGGER begins. If you are learning Assembly Language this program is a good example to use since you have the documented source code for it on the E/A disk. Get a print out of it to follow as you walk through the program with the Explorer.

When a program Auto-Starts it takes control right after it is loaded so the module does NOT: Load the color table - clear the screen - set up V7 - Load the User Workspace pointer to 20BA for the E/A or 70B8 for the Mini Mem and the Grom Address doesn't change.

Simulated Auto-Start Method

1. Set the Memory Pointer to c----s and using the Search Function in ASCII display find your program's Start Name (DEBUG) in the REF DEF Table. The start address for this table varies according to the number of DEFs in your program. For the the Editor/Assembler loader use 2600 for ST and 4000 for FN. For the Mini Memory loader use 7600 for ST and 8000 for FN.

2. When you have found the Start Name leave the Search Function and place the display in Hex mode. The structure of the REF DEF Table is: 6 Bytes for the name, including trailing space characters, followed by 2 Bytes that indicate the Start Address (PC). So for DEBUG the Hex display will be as follows:
   ```
   44 45 42 55 47 20 B0 BE
    D  E  B  U  G     .  .
   ```
   The first 6 bytes are the name followed by B0BE which is the Start Address (PC) for DEBUG.

3. Set the Cpu PC to B0BE and now you are ready to execute the DEBUGGER as if it was an Auto-Start program.

GPLLNK And The Grom Address

TI has a minor problem with Auto-Start programs that use GPLLNK (Grom subroutines). Right after a GPLLNK is executed the program Returns back to the Module instead of continuing on. If you program in Assembly you may have run into this at one time or another. We had you write down the Grom addresses in the first example to show you how to overcome this problem.

As you noticed a program that does not Auto-Start changes the Grom address by +>63 bytes. So, to overcome the problem in your programs just read the current Grom address when your program starts up, add >63 to it and then reset the Grom address to this new address. By adding >63 to the current address instead of just setting the address you will not have to worry about which module (E/A or MM) loaded it. This is only important if your program Auto-Starts and uses GPLLNK (Grom subroutines). This also applies to the Explorer when you use the Simulated Auto-Start Method. So when you use this method MAKE SURE you change the Grom address to the new address for your module. Here are the Grom addresses we have found, but they may be different in your modules.

Editor Assembler
>682F - Auto-Start - add >63 to it = >6892 - Non-Auto-Start address
Mini Memory
>68B3 - Auto-Start - add >63 to it = >6916 - Non-Auto-Start address

DIRECT EXECUTION OF MODULES

If you own a Navarone Widgit or a Myarc or Corcomp Disk Controller you can use the Explorer to execute modules other than Extended Basic, Editor Assembler or the Mini Memory. With one of these Disk Controllers you can have ANY module plugged into the cartridge port and use their Assembly Language loader utilities (CALL LR for Myarc or the Load And Run Assembly File from the Corcomp Disk manager). If you own a Widgit you can load the Explorer and then select one of the other modules to Explore. However, we have found that the Widgit has a nasty habit of SPIKING memory when the switch is moved so this is NOT very reliable. When it does spike memory, the Explorer usually goes out to lunch either right away or shortly after you start it up.

Once you have the Explorer loaded and the module you want to Explore on line you can start executing it in two different ways. The first, and longest, way is to go through the Power Up routine and then select the module from the menu. The second way is to do a quick SEARCH, set the address and begin executing the module. Since we have already discussed the Power Up routine we will Explore the second method. For this example lets Explore the Disk Manager 2 Command Module.

Non-Auto-Start Grom Modules

1. Set the Memory Pointer to g----- and press FCTN 5 - Search - and set ST to 6000 and FN to F7FF (Grom module space) and then press Enter or the down arrow to put the cursor in the Search string input field in ASCII mode with Basic Bias Off.

2. Type in as much of the module's name as you can. This is the name that appears on the menu. For some modules this may just be ENGLISH for others its the name. Include punctuation where its needed (ie: quotes or /). For our example we will type in "DISK MANAGE - don't forget the quote or the space between DISK and MANAGER and press Enter (the cursor is already on the last character of the search string).

3. When the cursor comes back and the search string has been found press FCTN 5 to turn of the Search Function. Next place the cursor in the Memory Window (edit) by pressing FCTN 9.

4. With cursor sitting in the home position (first byte) of the Memory Window press the left arrow 5 times and then press FCTN = to display the Hex values. This is the start of an Application Program Header and it has the following format:
   first word = 0000 = next Application Program Header - no more
   second word = 8134 = Entry point for this Application Program
   next byte = 0E = Name length for this Application Program
   next bytes = 2244 etc.= "DISK MANAGER"
   Make a note of the Entry Point address.

5. Move your cursor to the Grom AD (Address) field and input the Entry Point address (8134), make sure the Cpu WS is at 83E0 and the PC is at 006A. You are now ready to begin direct execution of the DISK MANAGER 2 Command Module.

Auto-Start Grom Modules (AAFF)

Auto-Start Grom modules are the ones that do not have a name on the Menu they just start right up. Many of the Scott Forsesman modules are of this type. Since they do not have an Application Program header you use a different method to directly execute these modules.

1. With Explorer loaded and the module on line set the Memory Pointer to g6000- and look at the first word in Hex. It should be AA FF. The AA is the validation flag for Grom Headers. If its not at this Grom space check 8000, A000, C000 and E000 until you find it.

   The second byte is the version number. For most Auto-Start Grom Modules this will be FF. The FF is version number -1 in two's compliment. When a module has a negative version number it signifies that it has a Foreign Language Translation routine in it. Scott Forsesman uses this routine to take control and start up their modules.

2. The Foreign Language Translation routine starts at 6013 in the Grom Module space (by default). So just set your Grom AD field to 6013, make sure the Cpu WS is at 83E0 and the PC is at 006A and start it up.

Auto-Start Grom Modules (Power Up Routine)

If a module isn't using the Foreign Language Translation routine it is possible that it is taking control with its Power Up routine. A standard Grom Module Header can reside at g6000, g8000, gA000, gC000 or gE000 and it has the following format:

```
>x000 = >AA Valid GROM Header Identification Code
>x001 = >00 Version number
>x002 = >00 Number of Application Programs
>x003 = >00 Reserved - not used
>x004 = >0000 Address of Power Up Header
>x006 = >0000 Address of Application Program Header
>x008 = >0000 Address of DSR Routine Header
>x00A = >0000 Address of Subprogram Header
>x00C = >0000 Address of Interrupt Routine Header - none in GROM
>x00E = >0000 Reserved for future expansion.
```

Replace the x (>x004) with the proper address (6,8,A,C or E). This header format is also used for Rom modules (Cpu memory 6000-7FFF) and DSR headers (Cpu memory 4000-5FFF with the DSR enabled)

1. Make a note of the value in gx004 - the Power Up Header address, if it is not zero. Set the Memory pointer to this address and make a note of the second word. This is the start address for the Power Up routine.

2. Set the Grom address to this second word, make sure the Cpu WS is at 83E0 and the PC is at 006A and start it up.

Non-Auto-Start Rom Modules

These modules are ones that were produced by some of the third party modules companies such as Atari Soft that generate a module name in the Menu. The procedure for direct execution of these modules is as follows:

1. Set the Memory Pointer to c----- and press FCTN 5 - Search - and set ST to 6000 and FN to 7FFF (Cartridge Rom module space) and then press Enter or the down arrow to put the cursor in the Search string input field in ASCII mode with Basic Bias Off.

2. Type in as much of the module's name as you can, place your cursor on top of the last character in the search string and press Enter

3. When the cursor comes back and the search string has been found press FCTN 5 to turn of the Search Function. Next place the cursor in the Memory Window (edit) by pressing FCTN 9.

4. With cursor sitting in the home position (first byte) of the Memory Window press the left arrow 5 times and then press FCTN = to display the Hex values. This is the start of an Application Program Header and it has the following format:

```
first word  = xxxx      = Next Application Program Header
second word = xxxx      = Entry point for this Application Program
next byte   =   xx      = Name length for this Application Program
next bytes  = xxxx etc.= (module's menu name)
```

   Make a note of the Entry Point address (second word)

5. Move your cursor to the Cpu PC (Program Counter) field and input the Entry Point address, make sure the Cpu WS is at 83E0 and you are now ready to begin direct execution of a Non-Auto-Start Rom Cartridge.

Auto-Start Rom Module

Most of these start up with the Power Up Routine, so:

1. Set the Memory Pointer to c6004-, make note of the first word (Power Up Routine Header address), set the Memory Pointer to this address and make note of the second word (Entry Point for the Power Up Routine).

2. Move your cursor to the Cpu PC field and input the Entry Point, make sure the Cpu WS is at 83E0 and you are now ready to begin execution of the Auto-Start Rom Cartridge.

A Few Notes On Modules

1. If you hop directly into a module or TI Basic AFTER you have been playing around with other items it is possible that:

   A. The character set may not be right.
   B. The Vdp registers are no longer set to the Power Up settings (see Appendix)
   C. A copy of Vdp Register 1 is NOT in >83D4. This may cause your screen to go blank when you press a key. The Key scan routine resets V1 with whatever is in the most significant byte of >83D4 when a key is pressed.

2. Some modules PRESUME that they are being executed right after the Power Up Routine so they fail to clear the screen or set up colors and Vdp Registers. If this occurs you will either have to set these items yourself or go through the Power Up Routine to execute the module properly.

3. MANY Modules use Auto-Sound processing and Auto-Sprite motion that is generated by the Interrupt Routine so interrupts will need to be enabled for proper execution.

4. MANY Grom Modules use the Vdp Interrupt Timer at >8379 as a delay timer to compare values to before continuing. If R3 contains >8379 and you don't seem to be getting anywhere then either: Change the value at >8379 to one greater than the one that keeps showing up in R0 or enable interrupts so this counter increments itself.

5. Some modules wait for >83CE to be zero before they continue (Auto sound processing completed). If R3 contains >83CE just press CTRL 5, this will turn off the sound and zero out >83CE for you.

6. Some modules, like TI LOGO, want to use the memory space that Explorer occupies and as such they will not execute properly.

   Other then these few items the execution of modules is very similar to executing Basic or Assembly Language programs. So have fun and DON'T stay up too late!

to make XB write any value to [illegible]

OVERALL SYSTEM MAP

```
+-----------------------------------------------------------------------+
|>0000    |CONSOLE ROM    Interrupt Vectors, XOP Vectors,       8K Bytes  |
|         |      GPL Interpreter, Floating Point Routines,                |
|>1FFF    |      XMLLNK Vectors, Low-level cassette DSR etc.              |
+-----------------------------------------------------------------------+
|>2000    |LOW MEMORY EXPANSION  RAM                            8K Bytes  |
|         |      Varies according to the loader used (Assembly).          |
|>3FFF    |      Generally not used by Extended Basic programs.           |
+-----------------------------------------------------------------------+
|>4000    |DSR ROM  Device service routines                     8K Bytes  |
|         |      Determined by CRU bit setting                            |
|>5FFF    |      Disk Controller, RS232, P-Code etc.                      |
+-----------------------------------------------------------------------+
|>6000    |CARTRIDGE PORT  ROM (& Mini-Mem RAM)                 8K Bytes  |
|         |      12K of Extended BASIC ROM. Upper 4K @ >7000 - >7FFF      |
|>7FFF    |      is flipped to page in another 4K for a total of 12K      |
+-----------------------------------------------------------------------+
|>8000    |RAM MEMORY MAPPED DEVICES VDP, GROM, SOUND & SPEECH   8K Bytes |
|  >8000  |      duplication of scratch pad ram @ >8300 ->83FF            |
|  >80FF  |                                                               |
|  >8100  |      duplication of scratch pad ram @ >8300 ->83FF            |
|  >81FF  |                                                               |
|  >8200  |      duplication of scratch pad ram @ >8300 ->83FF            |
|  >82FF  |---------------------------------------------------------------|
|  >8300  |  CPU SCRATCH PAD RAM        256 bytes                         |
|  >83FF  |---------------------------------------------------------------|
|  >8400  |  SOUND CHIP                                                   |
|  >87FF  |---------------------------------------------------------------|
|  >8800  |  VDP READ DATA                                                |
|  >8802  |  VDP STATUS (MSBy)                                            |
|  >8BFF  |                                                               |
|  >8C00  |  VDP WRITE DATA                                               |
|  >8C02  |  VDP READ/WRITE ADDRESS (to write set MSb of the MSBy to 01)  |
|  >8FFF  |---------------------------------------------------------------|
|  >9000  |  SPEECH READ                                                  |
|  >93FF  |                                                               |
|  >9400  |  SPEECH WRITE                                                 |
|  >97FF  |---------------------------------------------------------------|
|  >9800  |  GROM/GRAM READ DATA                                          |
|  >9802  |  GROM/GRAM READ ADDRESS                                       |
|  >9BFF  |                                                               |
|  >9C00  |  GROM/GRAM WRITE DATA                                         |
|  >9C02  |  GROM/GRAM WRITE ADDRESS                                      |
|>9FFF    |                                                               |
+-----------------------------------------------------------------------+
|>A000    |HIGH MEMORY EXPANSION  RAM                           24K Bytes |
|         |                                                               |
|         |        Extended Basic High Memory Usage, Free space end       |
|         |            pointed to by CPU RAM PAD address >8386            |
|         |                                                               |
|         |                     Numeric values                            |
|         |      ------------------------------------------------         |
|         |                     Line number table                         |
|         |      ------------------------------------------------         |
|         |                     X-Basic program space                     |
|         |                                                               |
|>FFFF    |                                                               |
+-----------------------------------------------------------------------+
```

CONSOLE ROM MEMORY MAP (most consoles)

>0000	Power Up Routine Vector - Level 0 Interrupt (Reset)
>0004	Level 1 - 9901 Interrupt Vector
>0008	Unused interrupt vector - points to screen timeout routine
>000C	Cpu clock speed for Baud rate generation and 'AA' for Validation
>000E	Assembly Language Key Scan Entry Point (B @>02B2) - subtract 4
	from the branch address to get the Gpl entry point.
>0012	Data - zero and eight
>0014	Instructions for the unreleased TI Debugger board
>0020	Branch and Link statement for BREAKPOINT check
>0022	Entry address of Breakpoint for RS232 and Basic
	Subtract >10 from it to get the start of the key range check
	Subtract >1A from it to get the start of the key debounce
	Subtract >1C from it to get the end of the key routine
>0024	**POWER UP ROUTINE** - start of console reset routine
>0038	More instructions for the unreleased TI Debugger board
>0040	XOP 0 vector - used by the TI Debugger board
>0044	XOP 1 vector - user defined - Not supported on early consoles
>0048	XOP 2 vector - user defined -
>004E	**START OF GPL INTERPRETER**
>0060	Set Grom/Gram address
>0070	Next Gpl instruction
>0082	= >0C36 Points to start of branch tables for Gpl instructions
>0C36	**GPL BRANCH VECTOR TABLES**
>0C36	0270 - Vector for Gpl miscellaneous instruction executor
>0C38	061E - Vector for >20 - MOVE instruction
>0C3A	011A - Vector for >40 - BRANCH ON RESET instruction
>0C3C	010E - Vector for >60 - BRANCH ON SET instruction
	Miscellaneous Instructions
>0C3E	0838 - Vector for >00 - RETURN instruction
>0C40	083E - Vector for >01 - RETURN WITH CONDITION BIT instruction
>0C42	027A - Vector for >02 - RANDOM instruction
>0C44	02AE - Vector for >03 - SCAN (Key Scan) instruction
>0C46	029E - Vector for >04 - BACK (Screen Color) instruction
>0C48	0104 - Vector for >05 - BRANCH instruction
>0C4A	085A - Vector for >06 - CALL instruction
>0C4C	05A2 - Vector for >07 - ALL (Clear Screen) instruction
>0C4E	04DE - Vector for >08 - FORMATTED BLOCK MOVE instruction
>0C50	00F4 - Vector for >09 - HIGH instruction
>0C52	00F4 - Vector for >0A - GREATER THAN instruction
>0C54	0024 - Vector for >0B - EXIT (Power Up) instruction
>0C56	00F4 - Vector for >0C - CARRY instruction
>0C58	00F4 - Vector for >0D - OVERFLOW instruction
>0C5A	18C8 - Vector for >0E - PARSE (Basic) instruction
>0C5C	0608 - Vector for >0F - XML instruction
>0C5E	1920 - Vector for >10 - CONTINUE (Basic) instruction
>0C60	1968 - Vector for >11 - EXECUTE (Basic) instruction
>0C62	19F0 - Vector for >12 - RETRUN TO BASIC (Basic) instruction
>0C64	082C - Vector for >13 - Unlisted - returns the Grom Base address
	from a library / program call
>0C66	
thru	>0C0C - Vector for >14->1E - Unlisted - points to routines for
>0C7A	the TI Debugger Board
>0C7C	>0C14 - Vector for >1F - Unlisted - points to routine for the
	TI Debugger board - part of
	its Breakpoint routine.

CONSOLE ROM MEMORY MAP Continued

>0C7E	Gpl instructions with a negative opcode (Greater than >7F)
>0C7E	>0136 - Vector for >80 - ABSOLUTE instruction
>0C80	>013A - Vector for >82 - NEGATE instruction
>0C82	>0140 - Vector for >84 - INVERT instruction
>0C84	>013E - Vector for >86 - CLEAR instruction
>0C86	>0144 - Vector for >88 - FETCH instruction
>0C88	>0162 - Vector for >8A - CASE instruction
>0C8A	>016E - Vector for >8C - PUSH instruction
>0C8C	>00EA - Vector for >8E - COMPARE ZERO instruction
>0C8E	>0186 - Vector for >90 - INCREMENT instruction
>0C90	>0188 - Vector for >92 - DECREMENT instruction
>0C92	>0184 - Vector for >94 - INCREMENT BY TWO instruction
>0C94	>0182 - Vector for >96 - DECREMENT BY TWO instruction
>0C96	>0C0C - Vector for >98 - Unlisted (TI Debugger board)
>0C98	>0C0C - Vector for >9A - Unlisted (TI Debugger board)
>0C9A	>0C0C - Vector for >9C - Unlisted (TI Debugger board)
>0C9C	>0C0C - Vector for >9E - Unlisted (TI Debugger board)
>0C9E	>0188 - Vector for >A0 - ADD instruction
>0CA0	>0186 - Vector for >A4 - SUBTRACT instruction
>0CA2	>01CE - Vector for >AB - MULTIPLY instruction
>0CA4	>01EA - Vector for >AC - DIVIDE instruction
>0CA6	>0190 - Vector for >B0 - AND instruction
>0CA8	>0196 - Vector for >B4 - OR instruction
>0CAA	>019A - Vector for >B8 - XOR instruction
>0CAC	>019E - Vector for >BC - STORE instruction
>0CAE	>01A2 - Vector for >C0 - EXCHANGE instruction
>0CB0	>00D6 - Vector for >C4 - COMPARE HIGH instruction
>0CB2	>00DA - Vector for >C8 - COMPARE HIGH OR EQUAL instruction
>0CB4	>00DE - Vector for >CC - COMPARE GREATER THAN instruction
>0CB6	>00CC - Vector for >D0 - COMPARE GREATER THAN OR EQUAL
>0CB8	>00EC - Vector for >D4 - COMPARE EQUAL instruction
>0CBA	>00E2 - Vector for >D8 - COMPARE LOGICAL
>0CBC	>01B0 - Vector for >DC - SHIFT RIGHT ARITHMETIC instruction
>0CBE	>01B4 - Vector for >E0 - SHIFT LEFT LOGICAL instruction
>0CC0	>01BB - Vector for >E4 - SHIFT RIGHT LOGICAL instruction
>0CC2	>01C2 - Vector for >E8 - SHIFT RIGHT CIRCULAR instruction
>0CC4	>06D2 - Vector for >ED - COINCIDENCE instruction
>0CC6	>0C0C - Vector for >F0 - Unlisted
>0CC8	>05C8 - Vector for >F6 - INPUT/OUTPUT instruction
>0CCA	>004E - Vector for >F8 - Unlisted (saves Grom address and base)
>0CCC	>0C0C - Vector for >FC - Unlisted
	MOVE INSTRUCTION VECTORS (Block Move >20->3F)
	These tables are used by the MOVE instructions.
	They move blocks of memory (Data) from a designated source
	to a designated destination.
	The first three entries move bytes from the source.
	The next four entries move bytes to the destination.
>0CCE	>0660 - CPU SOURCE
>0CD0	>0672 - GROM SOURCE
>0CD2	>0664 - VDP SOURCE
>0CD4	>0682 - CPU DESTINATION
>0CD6	>0686 - GRAM DESTINATION
>0CD8	>06BA - VDP DESTINATION
>0CDA	>0698 - VDP REGISTER DESTINATION

CONSOLE ROM MEMORY MAP Continued

>0CDC	**FORMAT INSTRUCTION TABLE** - The Formatted Block Move instruction is a sub interpreter within the Gpl interpreter.
>0CDC	>050A - Vector for STRING ACROSS
>0CDE	>0508 - Vector for STRING DOWN
>0CE0	>0504 - Vector for REPEAT ACROSS
>0CE2	>0502 - Vector for REPEAT DOWN
>0CE4	>0534 - Vector for SKIP ACROSS
>0CE6	>0532 - Vector for SKIP DOWN
>0CE8	>053A - Vector for REPEAT BLOCK
>0CEA	>056C - Vector for SPECIAL - write color table or loads XPT, YPT
>0CEB	**INPUT/OUTPUT INSTRUCTION TABLE**
>0CEB	>05D6 - Vector for EXECUTE SOUND LIST
>0CEE	>05D6 - Vector for EXECUTE SOUND LIST
>0CF0	>05E8 - Vector for CRU BIT INPUT
>0CF2	>05EA - Vector for CRU BIT OUTPUT
>0CF4	>1346 - Vector for CASSETTE WRITE ROUTINE (Low Level)
>0CF6	>142E - Vector for CASSETTE READ ROUTINE (Low Level)
>0CF8	>1426 - Vector for CASSETTE VERIFY ROUTINE (Low Level)
>0CFA	**LOCATION OF ALL 16 XML TABLES** (Vectors)
>0CFA	>0D1A - Vector for - >00 - Floating Point Table
>0CFC	>12A0 - Vector for - >10 - Pointer to XTAB
>0CFE	>2000 - Vector for - >20 *
>0D00	>3FC0 - Vector for - >30 * The Rest through >8300 are
>0D02	>3FE0 - Vector for - >40 * pointers to the other XML
>0D04	>4010 - Vector for - >50 * Tables
>0D06	>4030 - Vector for - >60 *
>0D08	>6010 - Vector for - >70 *
>0D0A	>6030 - Vector for - >80 *
>0D0C	>7000 - Vector for - >90 *
>0D0E	>8000 - Vector for - >A0 *
>0D10	>A000 - Vector for - >B0 *
>0D12	>B000 - Vector for - >C0 *
>0D14	>C000 - Vector for - >D0 *
>0D16	>D000 - Vector for - >E0 *
>0D18	>8300 - Vector for - >F0 *
>0D1A	**FLOATING POINT ROUTINES TABLE**
>0D1A	>0000 -
>0D1C	>0F54 - Vector for - ROUND Check to see if FAC needs rounding
>0D1E	>0FB2 - Vector for - ROUNU Round FAC starting at digit specified in ARG.
>0D20	>0FA4 - Vector for - STEXIT Store status
>0D22	>0FC2 - Vector for - OVEXP Over/underflow
>0D24	>0FCC - Vector for - OV Part of OVEXP
>0D26	>0D80 - Vector for - FADD Floating point add
>0D28	>0D7C - Vector for - FSUB Floating point subtract
>0D2A	>0E88 - Vector for - FMULT Floating point multiply
>0D2C	>0FF4 - Vector for - FDIV Floating point divide
>0D2E	>0D3A - Vector for - FCOMP Floating point compare
>0D30	>0D84 - Vector for - SADD Stack add
>0D32	>0D74 - Vector for - SSUB Stack subtract
>0D34	>0E8C - Vector for - SMULT Stack multiply
>0D36	>0FF8 - Vector for - SDIV Stack divide
>0D38	>0D46 - Vector for - SCOMP Stack compare

>12A0	**XTAB XML TABLE**		
>12A0	>11AE - Vector for -	CSN	Convert ASCII to floating point
>12A2	>11A2 - Vector for -	CSNGR	Grom entry for CSN routine
>12A4	>12B8 - Vector for -	CFI	Convert floating point to integer
>12A6	>1648 - Vector for -	SYM	Fetch BASIC symbol table entry
>12A8	>164E - Vector for -	SMB	Fetch BASIC symbol table value
>12AA	>1642 - Vector for -	ASSGNV	Assign BASIC variable
>12AC	>15D6 - Vector for -	SCHSYM	Search BASIC symbol table
>12AE	>163C - Vector for -	VPUSH	Push value onto VDP stack
>12B0	>1F2E - Vector for -	VPOP	Pop value from VDP stack
>12B2	>0AC0 - Vector for -	SROM	Search ROM. Part of DSR routine
>12B4	>0B24 - Vector for -	SGROM	Search GROM. Part of DSR routine
>12B6	>1868 - Vector for -	PGMCH	Get BASIC program character
>1C9C	**BASIC STATEMENT TABLE** - Used by EXEC (Execute)		
>1C9C	>1A2C - Vector for -	Spare	
>1C9E	>1A2C - Vector for -	ELSE	
>1CA0	>1A2C - Vector for -	Reserved : :	
>1CA2	>1A2C - Vector for -	Reserved !	
>1CA4	>1BB6 - Vector for -	IF	
>1CA6	>1A8E - Vector for -	GO	
>1CA8	>1AFC - Vector for -	GOTO	
>1CAA	>1AE0 - Vector for -	GOSUB	
>1CAC	>1B74 - Vector for -	RETURN	
>1CAE	>19E6 - Vector for -	DEF	
>1CB0	>19E6 - Vector for -	DIM	
>1CB2	>1A3C - Vector for -	END	
>1CB4	>8000 - Vector for -	FOR	
>1CB6	>1BEA - Vector for -	LET	
>1CB8	>8002 - Vector for -	BREAK	
>1CBA	>8004 - Vector for -	UNBREAK	
>1CBC	>8006 - Vector for -	TRACE	
>1CBE	>8008 - Vector for -	UNTRACE	
>1CC0	>8016 - Vector for -	INPUT	
>1CC2	>19E6 - Vector for -	DATA	
>1CC4	>8012 - Vector for -	RESTORE	
>1CC6	>8014 - Vector for -	RANDOMIZE	
>1CC8	>1C14 - Vector for -	NEXT	
>1CCA	>800A - Vector for -	READ	
>1CCC	>1A3C - Vector for -	STOP	
>1CCE	>803E - Vector for -	DELETE	
>1CD0	>19E6 - Vector for -	REM	
>1CD2	>1A92 - Vector for -	ON	
>1CD4	>800C - Vector for -	PRINT	
>1CD6	>800E - Vector for -	CALL	
>1CD8	>19E6 - Vector for -	OPTION	
>1CDA	>8018 - Vector for -	OPEN	
>1CDC	>801A - Vector for -	CLOSE	
>1CDE	>1A2C - Vector for -	SUB	
>1CE0	>803C - Vector for -	DISPLAY	
>1CE2	**TABLE USED BY PARSE**		
>1CE2	>801C - Vector for -	(	
>1CE4	>1A2C - Vector for -	&	
>1CE6	>1A2C - Vector for -	Spare	
>1CE8	>1A2C - Vector for -	OR	
>1CEA	>1A2C - Vector for -	AND	
>1CEC	>1A2C - Vector for -	XOR	
>1CEE	>1A2C - Vector for -	NOT	

CONSOLE ROM MEMORY MAP Continued

>1CF0	**LED TABLE USED BY PARSE**
>1CF0	>1A2C - Vector for - =
>1CF2	>1A2C - Vector for - <
>1CF4	>1A2C - Vector for - >
>1CF6	>801E - Vector for - +
>1CF8	>8020 - Vector for - -
>1CFA	>1A2C - Vector for - *
>1CFC	>1A2C - Vector for - /
>1CFE	>1A2C - Vector for - ^
>1D00	>1A2C - Vector for - Spare
>1D02	>8010 - Vector for - Quoted string
>1D04	>1A5C - Vector for - Unquoted string (NUMERIC)
>1D06	>1A2C - Vector for - Line number
>1D08	>804A - Vector for - EOF
>1D0A	>8022 - Vector for - ABS
>1D0C	>8024 - Vector for - ATN
>1D0E	>8026 - Vector for - COS
>1D10	>8028 - Vector for - EXP
>1D12	>802A - Vector for - INT
>1D14	>802C - Vector for - LOG
>1D16	>802E - Vector for - SGN
>1D18	>8030 - Vector for - SIN
>1D1A	>8032 - Vector for - SQR
>1D1C	>8034 - Vector for - TAN
>1D1E	>8036 - Vector for - LEN
>1D20	>8038 - Vector for - CHR$
>1D22	>803A - Vector for - RND
>1D24	>8040 - Vector for - SEG$
>1D26	>8046 - Vector for - POS
>1D28	>8044 - Vector for - VAL
>1D2A	>8042 - Vector for - STR$
>1D2C	>8048 - Vector for - ASC
	TABLE USED BY CONT (Continue)
>1D2E	>1D5C - Vector for - =
>1D30	>1D3E - Vector for - <
>1D32	>1D4C - Vector for - >
>1D34	>1DEC - Vector for - +
>1D36	>1E18 - Vector for - -
>1D38	>1E24 - Vector for - *
>1D3A	>1E30 - Vector for - /
>1D3C	>1E3C - Vector for - ^
>1E9C	VPUSH for GPL
>1EAA	VPUSH (XML >17)
>1F2E	VPOP (XML >18)
	This map contains mostly Vectors (Entry Points) for the various routines. By watching the address in the PC field of the Explorer you can tell which of these routines is being executed. You can also use these vectors to set the Cpu Breakpoint (BP) to halt the Explorer on a given routine so you can Step through it.

256 BYTES OF SCRATCH PAD RAM - XB USE

>8300	**XB TEMPORARY STORAGE AREA**
	This area of Scratch Ram is used by X-Basic and Basic as a temporary holding area for the different routines.
>8300	temporary variable
>8302	temporary variable
>8304	temporary variable
>8306	temporary variable - Record Length on file access
>8308	temporary variable - Address of Sprite Atttribute List
>830A	temporary variable
>830C	temporary variable
>830E	temporary variable - increment value for Auto Num
>8310	temporary variable - used in CALL LINK parameter passing
>8312	temporary variable - used by CHAR type statements
>8314	temporary variable - copy of VDP reg 1 for some commands
>8316	temporary variable - DSR Link flag for some commands
>8318	**XB PERMINENT STORAGE AREA**
	This area of Scratch Ram is used for specific items by X-Basic
>8318	Used by LINK, LOAD & rtn control to Basic also String space bgn
>831A	Points to 1st free add in VDP RAM also String space end
>831C	Points to allocated str space - PAB Error - Temp string pointer
>831E	Start of current statement
>8320	Current Screen Address
>8322	Return error code from Assembly Language Code
>8324	VDP value stack base pointer
>8326	Return address from Assembly Language Code
>8328	NUD Table for Assembly Language Code.
>832A	Ending screen display pointer
>832C	Program text or token code pointer
>832E	Pointer to current line number in line number table
>8330	Start of Line number table pointer
>8332	End of Line number table pointer
>8334	Data pointer for read
>8336	Line number table pointer for read
>8338	Address of intrinsic Poly constants
>833A	Subprogram symbol table pointer
>833C	PAB address in VDP RAM (first link) PAB list
>833E	Symbol table pointer
>8340	VDP Ram free space pointer
>8342	Current char/token
>8344	Extended Basic Program RUN = 255 STOP = 0 (w/o 'READY')
>8345	Extended Basic System Flags
	Bit 0 1 = Auto-Num Bit 4 1 = Edit Mode
	1 1 = On Break Next 5 1 = On Warning Stop
	2 6 1 = On Warning Next
	3 1 = Trace 7
>8346	Crunch buffer destruction level
>8348	Last subprogram block on stack

256 BYTES OF SCRATCH PAD RAM Continued

Address	Usage	DSR usage
>834A	**FLOATING POINT and DSR** usage, 36 bytes	
>834A	FAC (Floating point accumulator)	PAB I/O OPCODE
>834B	for floating point routines	PAB FLAG/STATUS
>834C	this area holds a number in	PAB DATA BUFFER ADDRESS
>834E	radix 100 notation.	PAB LOGICAL REC LENGTH
>834F		PAB CHARACTER COUNT
>8350		PAB RECORD NUMBER
>8352		PAB SCREEN OFFSET
>8353		PAB OPTION LENGTH
>8354	FLOATING POINT ERROR CODE	PAB DEVICE LENGTH
>8356	SUBRTN POINTER / DSR's pnts to 1st char after PAB in VDP	
>8358		DSR
>835A		DSR
>835C	ARG (Floating point argument)	DSR
	and DSR usage	DSR
		DSR
>836C	FPERAD (float pnt err add in Grom ?)	DSR
>836D	Set to >08 for DSR call	DSR
>836E	INTERPRETER and FLOATING POINT GPL VALUE STACK POINTER	
>8370	HIGHEST AVAILABLE ADDRESS IN VDP RAM	
>8372	LSByte OF DATA STACK•POINTER = A0 = (>83A0)	
>8373	LSByte OF SUBROUTINE STACK POINTER = 80 = (>8380)	
>8374	KEYBOARD NUMBER TO BE SCANNED Default =0	
>8375	ASCII CODE DETECTED by SCAN routine also SGN for float/point	
>8376	JOYSTICK Y-STATUS by SCAN routine also EXP for float/point	
>8377	JOYSTICK X-STATUS by SCAN routine	
>8378	RANDOM NUMBER GENERATOR RND's >0 ->63 (0-99)	
>8379	VDP INTERRUPT TIMER >0 ->FF (0-255)	
>837A	HIGHEST SPRITE # IN AUTO-MOTION >0 ->20 (0-32)	
>837B	COPY OF VDP STATUS REGISTER	
>837C	GPL STATUS BYTE (Set to 0 for a DSR CALL) (>20 =Key Press)	
>837D	CHARACTER BUFFER BYTE to VDP RAM screen table	
>837E	POINTS TO THE CURRENT ROW on the screen	
>837F	POINTS TO THE CURRENT COLUMN on the screen	
>8380	**THE DEFAULT SUBROUTINE STACK (Used by GPL Routines)**	
>8380	Reserved For Basics interpreter	
>8382	Reserved For Basics interpreter	
>8384	Reserved Highest Address in Expansion Memory	
>8386	Reserved Highest Free Address in Mem-Expansion	
>8388	Reserved For the Basic interpreter Sub stack base	
>8389	Reserved For the Basic interpreter Exp-Mem Flag	
>838A	RETURN ADDRESS STACK FOR GROM SUBROUTINES	
	(current Grom Address pushed to top of stack during Key Scan)	
>839E		
>83A0	**THE DEFAULT DATA STACK (Used by GPL Routines)**	
	this area holds various information according to the GROM	
	routine being executed.	
>83BF		

256 BYTES OF SCRATCH PAD RAM Continued

>83C0	**INTERRUPT WORKSPACE REGISTERS**	
>83C0	R0	RANDOM NUMBER SEED 2 Bytes >0-FF >0-FF
>83C2	R1	Bit 0 1 = disable ALL of the following
		1 1 = disable Auto Sprite Motion
		2 1 = disable Auto Sound Processing
		3 1 = disable The QUIT Key
		Bits 4-15 not used
>83C4	R2	ISR HOOK - Start address of User Interrupt Routine
>83C6	R3	Reserved for Keyboard state and debounce info
>83C8	R4	Reserved for Keyboard state and debounce info
>83CA	R5	Reserved for Keyboard state and debounce info
>83CC	R6	Pointer to Sound Table - also see >83FD
>83CE	R7	Number of Sound Bytes for Auto Sound Processing (0100)
>83D0	R8	Varies (>0000 for Cassette DSR Link)
>83D2	R9	Varies
>83D4	R10	CONTENTS OF VDP REGISTER 1 (used for key scan)
>83D6	R11	SCREEN TIME OUT COUNTER (blanks when incremented to 0000)
>83D8	R12	RETURN ADDRESS SAVED BY THE SCAN ROUTINE (Old R11)
>83DA	R13	Return WS for context switch (RTWP)
>83DC	R14	Return PC for context switch (RTWP)
>83DE	R15	Return ST for context switch (RTWP)
>83E0	**GPL WORKSPACE REGISTERS** (ALL Registers used by GPL interpreter)	
>83E0	R0	Varies NOTE: R0 - R7, R11 and R12
>83E2	R1	Varies are modified by Key Scan
>83E4	R2	Varies
>83E6	R3	Varies
>83E8	R4	Varies
>83EA	R5	Varies - Used by Interrupt Routine
>83EC	R6	Varies - Used by Interrupt Routine
>83EE	R7	Varies - Used by Interrupt Routine
>83F0	R8	Cleared on Return from Interrupt Routine
>83F2	R9	GPL Interpreter use
>83F4	R10	GPL Interpreter use
>83F6	R11	RETURN ADDRESS for BL instruction and User Interrupt
>83F8	R12	Varies - Cru Base Address for key scan and DSRs
>83FA	R13	GROM/GRAM READ DATA port (9800)
>83FC	R14	STATUS FLAGS
		Bits 0 - 7 Control the cursor blink speed & Auto sound processing. The value in this byte increments the counter at >8379
>83FD		Bit 0 — 4 1 = 16K Vdp Ram
		1 — 5
		2 1 = Cass Interrupt Timer — 6 1 = Multi-Color mode
		3 1 = Cass Verify — 7 Sound table location 1 = VDP 0 = Grom/Gram
>83FE	R15	VDP WRITE ADDRESS port (8C02)

Extended Basic LOW MEMORY EXPANSION after CALL INIT

Address	Value	Description of address
>2000	>205A	XML link to name link routine pointer.
>2002	>24FA	First Free address in low mem-exp.
>2004	>4000	Last Free address in low mem-exp.
>2006	>AA55	Constant that indicates CALL INIT has been executed.
>2008	**UTILITY VECTOR TABLE** (ie: BLWP @KSCAN)	
>2008	>2038	Utility workspace pointer for BLWP @NUMASG
>200A	>2096	**NUMASG** Utility starting address.
>200C	>2038	Utility workspace pointer for BLWP @NUMREF
>200E	>217E	**NUMREF** Utility starting address.
>2010	>2038	Utility workspace pointer for BLWP @STRASG
>2012	>21E2	**STRASG** Utility starting address.
>2014	>2038	Utility workspace pointer for BLWP @STRREF
>2016	>234C	**STRREF** Utility starting address.
>2018	>2038	Utility workspace pointer for BLWP @XMLLNK
>201A	>2432	**XMLLNK** Utility starting address.
>201C	>2038	Utility workspace pointer for BLWP @KSCAN
>201E	>246E	**KSCAN** Utility starting address.
>2020	>2038	Utility workspace pointer for BLWP @VSBW
>2022	>2484	**VSBW** Utility starting address.
>2024	>2038	Utility workspace pointer for BLWP @VMBW
>2026	>2490	**VMBW** Utility starting address.
>2028	>2038	Utility workspace pointer for BLWP @VSBR
>202A	>249E	**VSBR** Utility starting address.
>202C	>2038	Utility workspace pointer for BLWP @VMBR
>202E	>24AA	**VMBR** Utility starting address.
>2030	>2038	Utility workspace pointer for BLWP @VWTR
>2032	>24B8	**VWTR** Utility starting address.
>2034	>2038	Utility workspace pointer for BLWP @ERR
>2036	>2090	**ERR** Utility starting address.
>2038	**UTILITY WORK SPACE STARTS HERE**	
	R0-R15	
>2057	End of work space	
>2058		
>205A	Start of XML link to name link routine.	
	(Finds the name in the REF/DEF Table)	
>2090	Start of ERR Routine.	(Return Error code to basic)
>2096	Start of NUMASG Routine.	(Numeric Assignment)
>217E	Start of NUMREF Routine.	(Numeric Reference)
>21E2	Start of STRASG Routine.	(String Assignment)
>234C	Start of STRREF Routine.	(String Reference)
>2432	Start of XMLLNK Routine.	(Link to system Utilities)
>246E	Start of KSCAN Routine.	(Keyboard Scan)
>2484	Start of VSBW Routine.	(VDP single byte write)
>2490	Start of VMBW Routine.	(VDP multiple byte write)
>249E	Start of VSBR Routine.	(VDP single byte read)
>24AA	Start of VMBR Routine.	(VDP multiple byte read)
>24B8	Start of VWTR Routine.	(Write to VDP register)
	(NOTE: No GPLLNK or DSRLNK in X-Basic CALL INIT)	
>24FA	First Free Address in Low Mem-Exp. pointed to by >2002	
	- continued next page -	

Extended Basic LOW MEMORY EXPANSION Continued

```
|        | *---------------------------------------------------------* |
|        | *       The REF/DEF Table resides at the end of           * |
|        | *    Low Memory Expansion. Each entry is 8 bytes long.    * |
|        | *    6 for the Name and 2 for the starting address.       * |
|        | *    CALL INIT in X-Basic leaves this space empty.        * |
|        | *---------------------------------------------------------* |
|  >3FF0 | DEF Name (CALL LINK or BLWP @) 6 characters.                |
|  >3FF6 | Start address of the above routine, 2 bytes.                |
|  >3FF8 | DEF Name (CALL LINK or BLWP @) 6 characters.                |
|  >3FFE | Start address of the above routine, 2 bytes.                |
|>3FFF   |END OF LOW MEMORY EXPANSION                                  |
+--------------------------------------------------------------------+
```

Extended Basic HIGH MEMORY EXPANSION usage

```
+--------------------------------------------------------------------+
|>A000   |START OF HIGH MEM-EXPANSION                                |
|        |     (If Mem-Exp is present then the value at >8389        |
|        |      will be >E7 while the program is running)            |
|        |                                                           |
|        |    ---------------------------------------------------    |
|        |    NUMERIC VALUE TABLE  (in RADIX 100 notation)           |
|        |                                                           |
|        |    Starting point of the Symbol table in VDP RAM is       |
|        |    pointed to by >833E while the program is running.      |
|        |    The Symbol table then points into the Numeric value    |
|        |    table for each of the variable names.                  |
|        |    ---------------------------------------------------    |
|        | Highest Free Address in Mem-Exp. pointed to by >8386      |
|        |    ---------------------------------------------------    |
|        |    LINE NUMBER TABLE - 4 Bytes per entry.                 |
|        |                                                           |
|        |    | Line # = 2 Bytes | Start Address of line = 2 bytes | |
|        |    Line numbers are always stored highest # to lowest #   |
|        |    Starting address of this table is pointed to by >8330  |
|        |    Ending   address of this table is pointed to by >8332  |
|        |    Current line number being referenced                   |
|        |                    in this table is pointed to by >832E   |
|        |    ---------------------------------------------------    |
|        |    PROGRAM SPACE (Last line entered is at the top)        |
|        |                                                           |
|        |    Start of program space = (value at >8332)+1            |
|        |    Programs reserved words have been converted to Token values |
|        |    and the line numbers are removed from the beginning of |
|        |    each line. The format for each line is as follows:     |
|        |                                                           |
|        |           1st Byte = Number of bytes for the line         |
|        |   Following Bytes = (Start Address) Actual line code with |
|        |                     Token values replacing reserved words. |
|        |          Last byte = >00                                  |
|        |                                                           |
|  >FFE7 |  Highest address to be used in Mem-Exp. pointed to by >8384 |
|        |                                                           |
|  >FFFC |  Workspace Pointer  for LOAD Interrupt  (non-maskable interrupt, |
|  >FFFE |  Start Address (PC) for LOAD Interrupt       not DSK1.LOAD) |
|>FFFF   |END OF HIGH MEMORY EXPANSION                               |
+--------------------------------------------------------------------+
```

OVERALL VDP MAPS WITH BASIC AND EDITOR ASSEMBLER

```
Addr   BASIC                                Addr   EDITOR ASSEMBLER
      +----------------------------+              +----------------------------+
>0000 | SCREEN IMAGE TABLE         |        >0000 | SCREEN IMAGE TABLE         |
      | Start CHAR PATTERN TABLE   |              | Default start of Sprite    |
      | +96 Offset (>60 Bias)      |              | Pattern Table              |
      |                            |              |                            |
>02FF | END SCREEN IMAGE           |        >02FF | End Screen Image           |
      +----------------------------+              +----------------------------+
>0300 | COLOR TABLE                |        >0300 | SPRITE ATTRIBUTE TABLE     |
>031F |                            |              |                            |
      +----------------------------+              |                            |
>0320 | CRUNCH BUFFER              |              |                            |
>036F |                            |              |                            |
      +----------------------------+              |                            |
>0370 | CHARACTER PATTERN TABLE    |              |                            |
      | +96 Offset (>60 Bias)      |        >037F |                            |
      |                            |              +----------------------------+
      | >03C0-03DF Vdp Roll Out    |        >0380 | COLOR TABLE                |
      |                            |              |                            |
      | 768+8*character number=    |              |                            |
      |  address in decimal        |              | >03C0-03DF Vdp Roll Out    |
      |                            |              | >03E0-045F Value Stack     |
>03F0 | Character number 30        |              |                            |
>03F8 | Character number 31        |              | >0400 = char >80 in        |
>0400 | Character number 32        |              |         sprite pattern     |
>0408 | Character number 33        |              |                            |
      |     etc.                   |              |                            |
      |                            |              |                            |
      |                            |              |                            |
      |                            |        >077F |                            |
      |                            |              +----------------------------+
      |                            |        >0780 | SPRITE MOTION TABLE        |
>07FF |                            |        >07FF |                            |
      +----------------------------+              +----------------------------+
>0800 | PABS       (Value Stack)   |        >0800 | CHARACTER PATTERN TABLE    |
      | STRINGS                    |              | Standard Chars at          |
      | SYMBOL TABLES              |              | >0900 - >0AFF              |
      | NUMERIC VALUES             |              |                            |
      | LINE NUMBER TABLE          |              |                            |
      | PROGRAM SPACE              |              | Also used for PABs         |
      |                            |              |                            |
      |                            |        >0FFF |                            |
      |                            |              +----------------------------+
      |                            |        >1000 | FREE SPACE                 |
      |                            |              |                            |
      |                            |              | Also used for Loader PAB   |
      |                            |              |                            |
      |                            |              |                            |
      |                            |              |                            |
      |                            |              |                            |
      |                            |              |                            |
>35D7 |                            |        >35D7 |                            |
      +----------------------------+              +----------------------------+
>35D8 |                            |        >35D8 |                            |
      | DISK FILE BUFFERS          |              | DISK FILE BUFFERS          |
>3FFF |                            |        >3FFF |                            |
      +----------------------------+              +----------------------------+
```

16K VDP RAM Extended Basic Use

>0000	**VDP SCREEN IMAGE TABLE** **768 Bytes**
	1 Byte per screen position. Character value offset by >60 (96)
	(Row-1)*32+Col=Address
	>02E2=New line Address
>02FF	
>0300	**SPRITE ATTRIBUTE TABLE** **112 Bytes**
	4 Bytes per sprite. (room for 28 sprites)
>036F	¦ vert pos-1 ¦ horz pos ¦ char #+96 ¦ early clock bit : color ¦
>0370	**EXTENDED BASIC SYSTEM AREA** **128 Bytes**
>0371	Auto Boot needed Flag
>0372	Line to start execution at
>0376	Saved symbol table 'GLOBAL' pointer
>0378	Used for CHR$
>0379	Sound Blocks
>0382	Saved Program pointer for continue and Text pointer for break
>0384	Saved Buffer Level for continue
>0386	Saved Expansion Memory for continue
>0388	Saved Value Stack pointer for continue
>038A	On-Error Line pointer
>038C	Edit Recall start address
>038E	Edit Recall end address
>0390	Used as temp storage place (FAC12)
>0392	Saved main symbol table pointer
>0394	Auto load temp for inside Error
>0396	Saved last Subprogram pointer for continue
>0398	Saved On-Warning/Break bits for continue
>039A	Temp to save subprogram table
>039C	Same as above . Used in SUBS
>039E	Merged temp for PAB Pointer
>03A0	Random Number generator seed 2
>03A5	Random Number generator seed 1
>03AA	Input temp for pointer to Prompt
>03AC	Accept temp pointer
>03AE	Try Again
>03B0	Pointer to standard string in VALIDATE
>03B2	Length of standard string in VALIDATE
>03B6	Size temp for record length. Also temp in Relocating Program
>03B7	Accept "TRY AGAIN" Flag
>03B8	Saved pointer in SIZE when "TRY AGAIN"
>03BA	Used as temp storage place (FAC10)
>03BC	Old top of memory for Relocating Program / Temp for INPUT
>03BE	New top of memory for Relocating Program
>03C0	Temp Roll Out Area 32 Bytes
	(part of scratch pad RAM is moved here for various operations)
>03DC	Floating point sign
>03EF	

16K VDP RAM Extended Basic Use Continued

```
+---------+-----------------------------------------------------------+
|>03F0    | PATTERN DESCRIPTOR TABLE                       912 Bytes |
|         | SPRITE DESCRIPTOR TABLE                                   |
|         |                                                           |
|         | 8 Bytes per character / 114 characters (30-143)           |
|         |                                                           |
|         | >03F0 = Char 30     (The Sprite Motion Table uses the memory|
|         | >03F8 = Char 31      space for character sets 15 & 16)    |
|         | >0400 = Char 32                                           |
|>077F    |                                                           |
|---------+-----------------------------------------------------------+
|>0780    | SPRITE MOTION TABLE                            128 Bytes  |
|         | 4 Bytes/Sprite                                            |
|         |                                                           |
|>07FF    | | vert velocity | horiz velocity | sys use | sys use |    |
|---------+-----------------------------------------------------------+
|>0800    | COLOR TABLE                                     32 Bytes  |
|         | 1 Byte/Character set                                      |
|         |                                                           |
|>081F    | | foreground color : background color |                 |
|---------+-----------------------------------------------------------+
|>0820    | CRUNCH BUFFER                                  160 Bytes  |
|         | This area is used while crunching ASCII into token codes. |
|>08BE    |                                                           |
|---------+-----------------------------------------------------------+
|>08C0    | EDIT/RECALL BUFFER                             152 Bytes  |
|         | This area holds the info you type in on the command line. |
|>0957    |                                                           |
|---------+-----------------------------------------------------------+
|>0958    | VALUE STACK (Default Base Pointed to by >8324)            |
|         |                                                           |
|         | Used by the ROM routines SADD, SSUB, SMUL, SDIV, SCOMP etc.|
|>0967    | Top of Stack Pointed to by >836E  (GOSUBS Stacked here)   |
|         | --------------------------------------------------------- |
|>0968    | The items in this area move according        11840 Bytes  |
|         | to the size of the crunched program.                      |
|         |                                                           |
|         | The SYMBOL TABLES are generated (except the PAB) during   |
|         | the Pre-Scan period after you type RUN. The strings are   |
|         | placed in memory when they are assigned (ie: A$="Hello")  |
|         |                                                           |
|         | Without Mem-Exp                  With Mem-Exp             |
|         |                                                           |
|         | STRINGS                          STRINGS                  |
|         | ----------------------------     ------------------------ |
|         | DYNAMIC SYMBOL TABLE & PABS      DYNAMIC SYMBOL TABLE & PABS|
|         | ----------------------------     ------------------------ |
|         | STATIC SYMBOL TABLE              STATIC SYMBOL TABLE      |
|         | ----------------------------     ------------------------ |
|         | LINE NUMBER TABLE                Numeric Values, Line Number|
|         | ----------------------------     Table and Program Space moved|
|         | PROGRAM SPACE                    to High Mem-Expansion    |
|         | (crunched program)                                        |
|         |                                                           |
|         | The Line Number Table and the Crunched Program are saved to|
|         | disk like they reside in memory for PROGRAM "Memory Image"|
|>37D7    | type files.                                               |
+---------+-----------------------------------------------------------+
```

>37D8	**DISK BUFFERING AREA for CALL FILES(3)**	**5 Bytes**
>37D8	Validation code for the Disk Controller DSR (>AA)	
>37D9	Points to TOP of VDP memory (>3FFF)	
>37DB	CRU base identification (11 for CRU 1100)	
>37DC	Maximum number of OPENed files (>03 default)	
	File Control Block for 1st file OPENed 6 Bytes	**518 Bytes**
>37DD	Current Logical record offset	
>37DF	Sector number location of File Descriptor Record	
>37E1	Logical Record Offset (only used with VARIABLE records)	
>37E2	Drive number (high order bit set = Updated Data Buffer area)	
	File Descriptor Record (brought in from the disk 256 Bytes)	
>37E3	File Name	
>37ED	Reserved (>0000)	
>37EF	File Status Flags (file type and write protection)	
>37F0	Max number of Records per Allocation Unit (1 AU = 1 Sector)	
>37F1	Number of Sectors currently allocated (256 byte blocks)	
>37F3	End of File offset within the last used sector	
>37F4	Logical record length (ie: FIXED 80 or VARIABLE 254 etc.)	
>37F5	# of FIXED length records or # of sectors for VARIABLE length (the bytes are reversed ie: LSB¦MSB should be MSB¦LSB)	
>37F7	Reserved (>0000 >0000 >0000 >0000)	
>37FF	Pointer Blocks - 6 nibble, 3 byte, clusters that point to the Start Sector numbers and the highest logical Record Offset in the cluster. Change the nibble order from ¦ss2:ss1¦ ¦ro1:ss3¦ ¦ro3:ro2¦ to ¦ss3:ss2:ss1¦ ¦ro3:ro2:ro1¦	
>38E3	Data Buffer area 256 Bytes	
>39E3	File Control Block for 2nd file OPENed 6 Bytes same pattern as above	**518 Bytes**
>39E9	File Descriptor Record 256 Bytes same pattern as above	
>3AE9	Data Buffer area 256 Bytes	
>3BE9	File Control Block for 3rd file OPENed 6 Bytes same pattern as above	**518 Bytes**
>3BEF	File Descriptor Record 256 Bytes same pattern as above	
>3CEF	Data Buffer area 256 Bytes	
>3DEF	**VDP STACK AREA**	**252 Bytes**
>3EEA	Used by the Disk Controller DSR	
	DISK DRIVE INFO	**4 Bytes**
>3EEB	Last Drive Number accessed	
>3EEC	Last track access on Drive 1	
>3EED	Last track access on Drive 2	
>3EEE	Last track access on Drive 3	
>3EEF >3EF4	(?? not used any more was for the 99/4 ??)	**6 Bytes**
>3EF5	**VOLUME INFORMATION BLOCK** (Copy of Sector 0 from the last disk accessed for a <u>WRITE</u>)	**256 Bytes**
>3FF4	Contains Disk Name, type and bit map for used sectors.	
>3FF5	**FILE NAME COMPARE BUFFER**	**11 Bytes**
>3FFF	Contains disk number and 10 character file name for last access.	

CONSOLE GROM CHIP 0 (Monitor)

>0000	**GROM HEADER**
>0000	>AA Valid GROM Header Identification Code
>0001	>02 Version number
>0002	>0000 Number of Programs. .... none here
>0004	>0000 Address of Power Up Header .... none here
>0006	>0000 Address of Application Program Header .... none here
>0008	>1310 Address of DSR Routine Header
>000A	>1320 Address of Subprogram Header
>000C	>0000 Address of Interrupt Link .... none in GROM
>000E	>0000 Reserved for future? expansion.
>0010	**GPLLNK SUBROUTINE VECTOR TABLE**
	The values in these tables contain the instruction >40 (BR) which is BRANCH if condition bit in status register is RESET and the address is relative to the 6K GROM chip it resides in. Actual address for GROM 0 = value - >4000 (ie: >43DC = >03DC)
>0010	>43DC LINK programs to link between programs and DSR's
>0012	>443C Return from LINK or DSR
>0014	>49A9 CNS - Convert number into a string
>0016	>4396 Load Title screen characters
>0018	>439E Load Regular upper case characters
>001A	>4446 Generate Basic WARNING message
>001C	>4449 Generate Basic ERROR message
>001E	>444C Begin execution of GROM Basic
>0020	>4052 GROM Power Up routine
>0022	>51FE INT - Convert floating point to Integer function
>0024	>4C82 ^ - Exponentiation, raise a number to a power
>0026	>4D59 SQR - Square Root function
>0028	>4DB4 EXP - Exponential function
>002A	>4E64 LOG - Natural Logarithm function
>002C	>4EF9 COS - Cosine function
>002E	>4F01 SIN - Sine function
>0030	>4F5F TAN - Tangent function
>0032	>4F80 ATN - Arctangent function
>0034	>43CE - Generate BEEP sound
>0036	>43D6 - Generate HONK sound
>0038	>054D12 = BRANCH to GROM 2 >4D12 Get String Space routine
>003B	>525E - Bit reversal routine
>003D	>4417 - Special GROM entry point for Cassette DSR, points to a GROM routine that calls an XML to execute the low level Cassette DSR in the console ROM which returns to the high level Cassette DSR in GROM.
>003F	>052844 = BRANCH to GROM 1 >2844 Memory space check for PAB's
>0042	>0537B4 = BRANCH to GROM 1 >37B4 GPL subprogram setup
>0045	>60 = DATA - Basics screen character offset
>0046	>0D00 = DATA - Speech Read address (>0D00 + >8300 = >9000)
>0048	>1100 = DATA - Speech Write address (>1100 + >8300 = >9400)
>004A	>43C2 Load Lower case characters
	--- The following three were changed in the later version of ---
	--- GROM - After approx 3/82 or LTA 1482 ---
>004C	>04B4 Address of the Title Screen character data table
>004E	>06B4 Address of the Regular upper case character data table
>0050	>0874 Address of the Lower case character data table

	GROM ROUTINES
>0052	POWER UP ROUTINE (displays the Title Screen)
>0396	LOAD TITLE SCREEN characters routine
>039E	LOAD REGULAR UPPER CASE characters routine
>03C2	LOAD LOWER CASE characters routine
>03CE	GENERATE BEEP sound routine
>03D6	GENERATE HONK sound routine
>03DC	LINK ROUTINES for linking between programs and DSR's
>043C	RETURN from link or DSR
>0446	>05284C = BRANCH to GROM 1 >284C - WARNING routine
>0449	>05284E = BRANCH to GROM 1 >284E - ERROR routine
>044C	>052010 = BRANCH to GROM 1 >2010 - Execute Basic
>044F	**DATA TABLES**
>044F	DATA >80 (Hex 80)
>0451	DATA for VDP Register default values
>0459	DATA for Color Table default values for Title Screen
>0479	DATA for BEEP sound
>0484	DATA for HONK sound
>048F	DATA :1981:
>0496	DATA :TEXAS INSTRUMENTS:
>04A7	DATA :HOME COMPUTER:
>04B4	DATA for Title Screen Characters (CHR$(32-95))
>06B4	DATA for Regular Upper Case Characters (CHR$(32-95))
>0874	DATA for Lower Case Characters (CHR$(96-126))
>094D	DATA :FOR:
>0950	DATA for TI LOGO loaded at CHR$(1) in the Pattern Desc. Table
>09A0	**FLOATING POINT ROUTINES**
>09A0	Roll Out routine- moves part of Scratch Pad to VDP Roll Out Area
>09A9	CNS - Convert Number into String routines
>0AE6	Roll In routine- moves VDP Roll Out Area back into Scratch Pad
>0AEF	Balance of CNS routines
>0C6C	V PUSH - Push a number from FAC onto the VDP Value Stack
>0C77	V POP - Pop a number off the VDP value Stack to FAC
>0C82	^ - Exponentiation, raise a number to a power
>0D59	SQR - Square Root function
>0DB4	EXP - Exponential function
>0E64	LOG - Natural Logarithm function
>0EF9	COS - Cosine function
>0F01	SIN - Sine function
>0F5F	TAN - Tangent function
>0F80	ATN - Arctangent function
>0FDB	DATA and misc constants used by the Floating Point routines
>117B	Misc subroutines used by the Floating Point routines
>11FE	INT - Integer function
>125E	BIT REVERSAL ROUTINE
>1267	DATA this is the >40 bytes that is moved into >8300 and used by the Bit reversal routine.
>12A5	DATA :REVIEW MODULE LIBRARY: (currently not used)
>12C0	/ \
>130F	\ Unused area contains >0000 /

	CASSETTE DSR - High Level - checks for OPEN errors, displays screen messages for cassette operation etc.
	PAB set up for DSR (see Editor/Assembler manual pages 291-304)
	PAB+0 - I/O Opcode (Open, Close, Load, Save etc.)
	PAB+1 - Flag/Status (File-type, Mode of Operation & Data-type)
	PAB+2 - VDP Data Buffer Address
	PAB+4 - Logical Record Length
	PAB+5 - Character Count (bytes) to be transferred
	PAB+6 - Record Number (0-32767 not used for cassette I/O)
	PAB+8 - Bias for ASCII characters (>60 in Basics)
	PAB+9 - Length of the Device Name (>03 for CS1)
	PAB+10 - Start of the Device Name 'CS1' or 'CS2'
>1310	**DSR Header(s)**
>1310	>1318 - Pointer to next Device Name Header
>1312	>1326 - Start address for this Device
>1314	>03 - Name length for this Device
>1315	>435331 - DATA :CS1:
>1318	>0000 - Pointer to next Device Name Header - no more
>131A	>132C - Start address for this Device
>131C	>03 - Name length for this device
>131D	>435332 - DATA :CS2:
>1320	**SUBPROGRAM Header**
>1320	>0000 - Pointer to next Subprogram header - no more
>1322	>1573 - Start address for this Subprogram
>1324	>01 - Name length for this subprogram
>1325	>03 - DATA :03: (can not CALL CTRL C (CHR$(3)) from Basics)
>1326	Start of CS1 DSR (set up for CS1)
>131A	Start of CS2 DSR (set up for CS2)
>1330	Both CS1 & CS2 come here to start DSR
>1374	DO CASE Branch table for OPEN, CLOSE, READ Record, WRITE Record, RESTORE/REWIND, LOAD, SAVE, DELETE(close)
>135D	ERROR and EXIT routines
>1387	**CASSETTE ROUTINES**
>1387	OPEN a file routine
>13CF	READ a Record routine
>13DA	WRITE a Record routine
>13DD	Transfer data routine for READ and WRITE
>13F2	LOAD a file routine
>140E	CLOSE a file routine
>1444	VERIFY cassette data routine
>1489	SAVE a file routine
>1499	**CASSETTE SUBROUTINES** - These subroutines display the messages on the screen for cassette operation, turn on/off the cassette motors, look for key presses and wait for the leader to pass.
>1549	Cassette Motor On ----- >155E Cassette Motor Off
>1562	Wait for leader to pass
>1573	SUBPROGRAM >03 - Adds Bias >60 to the Cassette messages
>15A0	DATA TABLES
>15A0	Cassette operation messages
>16E0	Joystick Codes for key scan
>1700	Small Character codes for key scan
>1730	Shift Table codes for key scan
>1760	FCTN Table codes for key scan
>1790	CTRL Table codes for key scan
>17C0	Table for modes 1 & 2 for key scan

CONSOLE GROM CHIP 1

>2000	**GROM HEADER**
>2000	>AA Valid GROM Header Identification Code
>2001	>02 Version number
>2002	>01 Number of Programs.
>2003	>00 Reserved
>2004	>0000 Address of Power Up Header .... none here
>2006	>214D Address of Application Program Header
>2008	>0000 Address of DSR Routine Header
>200A	>4D1A Address of Subprogram Header (in GROM Chip 2)
>200C	>0000 Address of Interrupt Link .... none in GROM
>200E	>0000 Reserved for future? expansion.
	GROM CHIP 1 VECTOR TABLE (>2000 offset)
>2010	>4417 Routine to begin execution of Basic program in GROM
>2012	>4195 Routine to clear flags & set up keyboard
>2014	>460B Routine to parse (scan) an inputted command line
>2016	>466C Routine to generate the SYNTAX ERROR message.
>2018	>467E Routine to restore cursor position after Error
>201A	>4192 Secondary entry point for Basic Interpreter
>201C	>47F1 Routine that CALLs routines in GROM 0 to load characters
>201E	>436D Routine to move blocks of VDP RAM
>2020	>46AB Routine to reset the length byte for strings and numerics
	ERROR MESSAGES DATA TABLE
>2022	The Error messages in this table have a >60 (96) offset added to them for Basic so use the Explorer's Basic Bias to see these. (1st byte=length - Next bytes=message)
	APPLICATION PROGRAM Header
>214D	>0000 Pointer to next Application Program Header ... none here
>214F	>216F Start address for this program (Main entry point)
>2151	>08 Name length for this program
>2152	>54492042 DATA :TI BASIC: (for the menu screen)
>2156	>41534943
>215A	>422B Vector for routine that erases the symbol table (>222B)
>215C	DATA for the cursor character pattern
>2164	DATA for the screen edge character pattern
>216C	DATA for VDP Registers 2, 3 and 4 (>F0 0C F8)
>216F	**START OF TI BASIC INTERPRETER**
	The input line is scanned for the entries at >2214 and branches to them. If not one of these it executes the direct command (ie: CALL CLEAR or PRINT B+C etc.).
>216F	Entry point for 'NEW' routine
>2192	Secondary entry point for Basic Interpreter
>2195	Routine to clear flags, set up keyboard & prepare for input on the command line.
>21D6	Edit Routine that CALLs other routines to store the input from the keyboard into the VDP RAM Screen Image Table.
>21E5	Routine that CALLs another routine to scan the line just input and convert it into token codes and store it in VDP RAM
>2214	CASE branch table for:
	RUN NEW CONTINUE
	LIST BYE NUMBER
	OLD RESEQUENCE SAVE and EXIT'

	TI BASIC INTERPRETER Cont.
>222B	Entry point for routine that erases the Symbol Table
>2245	Entry point for 'LIST' routine
>224D	Entry point for 'RUN' routine
>2268	Entry point for 'CONTINUE' routine
>228C	Entry point for 'NUMBER' routine
>229F	Entry point for 'SAVE' routine
>22A7	Entry point for 'OLD' routine
>22AA	Entry point for 'RESEQUENCE' routine
>2342	Entry point for 'BYE' routine
>236D	Routine to move blocks of VDP RAM from a Lower address to a Higher address as you input program lines. >8300 = VDP location to move FROM >8302 = VDP location to move TO >835C = Number of bytes to move
>2377	Entry point for 'EDIT' (program lines) routine
>2417	Routine to begin execution of Basic program
>2457	Routine to scan an inputted command line CALLed from >21E5
>266C	Routine to generate the SYNTAX ERROR message.
>267E	Routine to restore cursor position after Error
>26AB	Routine to reset the length byte for strings and numerics
>27E3	Routine that clears the screen, resets the cursor and edge characters and then executes the following routine
>27F1	Routine that CALLs routines in GROM 0 to load character sets, then it resets the foreground and background colors and resets VDP Registers 2, 3 and 4
>2828	**VECTOR TABLE FOR EDIT & PRESCAN ROUTINES** (>2000 offset)
>2828	>4FFF Prescan (builds symbol table and checks for errors)
>282A	>4F43 Generates Bad Line Number error message
>282C	>4C75 Routine to parse the input line for non space chars
>282E	>4DFA Lists a program line to screen (converts token code into ASCII, reserved words)
>2830	>4CA6 Gets a valid character from the input line
>2832	>4A42 Main edit routine to read in a line from the keyboard
>2834	>4C36 Starts auto number with our line # and increment.
>2836	>4FC4 Finds where the first token is stored in vdp ram for line
>2838	>4BD6 Deletes and inserts program lines (moves memory around)
>283A	>4F12 Checks for valid line number
>283C	>4EF9 Converts a line number from ASCII into Binary value
>283E	>4F5D Locates a program line in vdp ram
>2840	>4C2B Starts auto number with default values of 100,10
>2842	>4FAF Converts line # from Binary to ASCII and displays it
>2844	>5493 Checks for room for symbol table or pab, this routine may execute a garbage collection and try again
>2846	>5450 Checks for type of char 0-9 a-z A-Z etc.
>2848	>51E5 Places a variable in the symbol table
>284A	>522B Puts dummy entries into the symbol table
>284C	>4D24 Prints out the WARNING messages
>284E	>4D99 Prints out the ERROR messages
>2850	>4C84 Checks the GPL stack and moves a char into it
>2852	>4CA0 Increments the VDP pointer for the next char
>2854	>4CC0 Handles unquoted strings adds unquoted token & len to it
>2856	>4C7A Gets first non space char from the input line
>2858	>4A49 Secondary edit routine, allows different line length
>285A	>4A4F Third edit routine, allows different starting cursor pos

>285C	**RESERVED WORD TOKEN TABLE** First 10 words point to the start of reserved word groupings. Groups are broken up by number of characters (1-10) per reserved word. The Token value follows the reserved word
>2A42	**LINE EDITOR**
>2A42	Routine that accepts keystrokes into a screen line. This is a line editor, it knows Insert, Delete etc. This entry point sets the default starting point and line length for Basic
>2A49	Second entry point for the line editor. By setting the line length in >835E before branching here you can change the maximum line length
>2A4F	Third entry point for the line editor. By setting the line length in >835E and the start point in >8361 before branching here you can have your input start and stop any place on the screen
>2BD6	Routine that moves memory around for inserting and deleting program lines
>2C2B	Routine that sets up the values for NUMBER (auto line numbering)
>2C75	Routine that parses a line and gets the non space chars
>2C7A	Routine that gets the first non space char. Both this routine and the one above CALL the routine at >2CA6
>2C84	Routine that checks the stack and moves a char to it
>2CA0	Routine that increments the VDP pointer and jumps to >2C84
>2CA6	Routine that checks for strings or numerics and handles each
>2CC0	Routine that handles unquoted strings, adds token & length to it
>2D24	Routine that prints the WARNING message on the screen
>2D99	Routine that prints the ERROR message on the screen. The pointer to the length byte in GROM for the WARNING or ERROR message is in the Scratch Pad at >8376
>2DFA	Routine that lists a program line on the screen. Starting point for the line is in >8302
>2EF9	Routine that converts an ASCII line number into binary
>2F12	Routine that checks for valid line number input
>2F43	Routine that generates the BAD LINE NUMBER error message
>2F5D	Routine that finds a line from the line number table
>2FAF	Routine that converts line # from binary to ASCII & displays it
>2FC4	Routine that finds the first token of a program line in VDP
>2FFF	PreScan routine, scans line or program and builds symbol table
>31E5	Routine that places the variable in the symbol table
>3450	Routine that checks char type 0-9, a-z, A-Z etc. Character that is checked is at >8342. This routine sets the condition bit in the GPL Status register if char is valid for variable name
>3493	Routine that checks for enough room for a symbol table entry or a PAB. If there's not enough room between the symbol table and the string space it tries to move the string space to a lower address, this may execute a garbage collection. If there still isn't enough room it generates the MEMORY FULL error message. (Word at >834A = space needed in bytes)
	NOTE: Most of the above routines use the FAC and ARG sections of Scratch Pad RAM for parameter passing. Some of them will use the temporary space at >8300 - >8316. Usually whenever a a routine does anything with a single character the character is at >8342. Also, most of the references to Scratch Pad are with an offset of >8300. ie: opcode BF 14 0008 = Double byte store 0008 at >8314

	BRANCH TABLE FOR A FEW OF THE ERROR MESSAGES
>3510	>05 5671 = Branch to 5671 - ILLEGAL STATEMENT
>3513	>05 567D = Branch to 567D - MEMORY FULL
>3516	>05 4D7C = Branch to 4D7C - BAD VALUE
>3519	>05 4D81 = Branch to 4D81 - STRING-NUMBER MISMATCH
	ENTRY POINTS FOR A FEW OF THE CALL STATEMENTS
>351C	**CLEAR** - Places the space character + bias (>60) in every screen position by using the GPL statement of ALL : :
>3527	DATA for SOUND >42,>0B,>12,>22,>00,>00,>00,>00
>352F	DATA for SOUND >01,>FF,>01,>04,>9F,>BF,>DF,>FF,>00
>3538	**SOUND** - This routine handles the entire sound statement. First it checks the duration, then it converts it into 1/60 seconds because sounds are interrupt driven. Next it finds the first frequency and divides it into 111834 (111834/freq) and passes that value to a sound table it is setting up in VDP RAM. Next it gets the volume and sets that up and then passes all the values to the sound chip (>8400). Interrupt routine is in the console ROM chip.
>360E	**HCHAR** - This routine and the VCHAR routine both call a subroutine at >37D6 to parse the statement for X,Y,CHAR,#CHRS and converts these into integer values. Then it puts them on the screen using a FMT statement (Formatted block move) that allows for writing over the border characters.
>362A	**VCHAR** - This is very similar to the above statement except that it places the characters vertically. The number of characters is at >834A, the character is at >8300, screen row is at >837E and the screen column is at >837F.
>3643	**CHAR** - This routine converts the string into the proper values for defining a character and moves these values into VDP RAM at the proper character + bias (>60) location. Both FAC (>834A) and ARG (>835C) are heavily used during this CALL. This routine appears to set up a temporary string in VDP RAM so it is possible that it could invoke a garbage collection and if there isn't enough room it will generate a Memory Full error message.
>3708	**KEY** - This parses the statement for the key unit, checks it for the proper range, CALLs >3767 to move it to >8374 and then executes the SCAN routine. After returning it checks the Status and places the proper value into your variable. Next it evaluates the keycode, converts it into floating point and places it in your variable.
>3748	**JOYST** - This is very similar to the above statement except after returning from >3767 it computes the proper X and Y values by CALLing >5755 and then places them into your variables.
>3767	Subprograms to do parsing for left parenthesis and commas, range checking for a range of 1-16, >0 or a preselected range.
>378E	Subprogram to parse the row and column values out of a graphics statement (ie: CALL HCHAR...).
>37BF	**SCREEN** - This subprogram sets the Screen and border color. It uses the above subroutines to parse the statement and then places the value into VDP register 7.
>37D6	Subprogram to parse HCHAR and VCHAR statements for row, column (by CALLing >378E), ASCII character value and number of characters.

CONSOLE GROM CHIP 2

>4000	**VECTOR TABLE FOR FILE ROUTINES** (>0000 Offset)
>4000	>426C DISPLAY routine
>4002	>4160 DELETE routine
>4004	>4227 PRINT routine
>4006	>4344 INPUT routine
>4008	>401E OPEN routine
>400A	>4174 CLOSE routine
>400C	>41D7 RESTORE routine
>400E	>45E3 READ routine
>4010	>4956 GET DATA FROM GROM/GRAM or RAM
>4012	>41CF CLOSE ALL OPEN FILES routine
>4014	>46FC PROGRAM SAVE routine
>4016	>4641 PROGRAM LOAD routine
>4018	>474C LIST routine
>401A	>4BFC OUTPUT RECORD routine
>401C	>482B END OF FILE routine
>401E	**OPEN ROUTINE** - This handles OPEN #x:"device.xx",VARIABLE xx,...
>40AF	Case branch table for the following OPEN parameters:
>4051	>40AB VARIABLE
>4053	>406B RELATIVE
>4055	>40D1 INTERNAL
>4057	>4070 SEQUENTIAL
>4059	>4095 OUTPUT
>405B	>409A UPDATE
>405D	>40A4 APPEND
>405F	>40B0 FIXED
>4160	**DELETE ROUTINE** - This handles the various DELETE functions
>4174	**CLOSE ROUTINE** - This handles CLOSE #x or CLOSE #x:DELETE
>41CF	**CLOSE ALL FILES ROUTINE** - This closes all open files
>41D7	**RESTORE ROUTINE** - This handles RESTORE (data), RESTORE xx (data) RESTORE #x and RESTORE #x,REC x for files
>4227	**PRINT ROUTINE** - This handles both screen and file PRINT. Both this and the Display routine check for Internal or Display type records and handle each accordingly.
>426C	**DISPLAY** - This handles the screen DISPLAY statement (no files)
>4344	**INPUT ROUTINE** - This handles both the screen and file INPUT it also checks data type against variable type
>45E3	**READ ROUTINE** - This handles the reading of DATA into variables it is not used for files. CALL's routines at >48CC - >4992
>4641	**OLD ROUTINE** - This is the OLD DSK1.xxxxx or OLD CS1 routine, it sets up the PAB, Calls the DSR, Tests the Checksum, gets the new addresses for the end & start of the line # table, makes adjustments for different RAM size (4K?) and stores them at >8332 & >8330 respectively. Adjusts the memory and updates the line # pointers if different RAM size. Both OLD & SAVE CALL routines at >4888 - >48CB
>46FC	**SAVE ROUTINE** - This is SAVE DSK1.xxx or SAVE CS1, it closes all open files, clears all break points, stores the start and end pointers for the line # table, finds the number of bytes used (>8370), passes it to the PAB and calls the DSR for a SAVE.
>474C	**LIST ROUTINE** - This lists out the program lines to the screen or to the device specified. Unfortunately it generates a Syntax error if you use anything but a : after the device name. ie; LIST "PIO":100-150 is OK but not "PIO",VARIABLE 28
>482B	**END OF FILE ROUTINE** - This is the EOF(x) function.

	SUBROUTINES
>4888	**OLD & SAVE SUBROUTINE** - This gets the program name, initializes many of the program pointers, deletes the symbol table, sets up the PAB and returns.
>48CC	**READ & INPUT SUBROUTINES** - These find the symbol table entries, check for Strings or Numerics, decide if its GROM or RAM data and pass the Data item to the variable.
>4956	- GET DATA FROM GROM OR RAM - Reads the next Data item from GROM if the GROM Flag at >8389 is in >834D. If >834D is = 0 then the next Data item is read from RAM.
>4993	**OPEN, CLOSE & RESTORE SUBROUTINES** - These parse out the file number (ie: #1, if its there), check for the proper range (> 0 and < 256), scan the PAB chain for the proper file. If any of these items are not right it returns with an error. On a Close the routine at >49E6 deletes the PAB and adjusts the memory and PAB chain pointers.
>4B53	**PRINT SUBROUTINES** - These handle the outputting of data to the screen or to a file. They check for valid separators (,;:) and handle each accordingly. For screen output they add the character offset (>60) to each character.
>4BFC	- OUTPUT A RECORD - This is the subroutine that outputs a a record to either the screen or an output device, depending on the PAB (file #0 = screen output)
>4D00	**VECTOR TABLE FOR BASIC EXECUTION**
>4D00	>56CD Screen Scroll Routine
>4D02	>5120 Move a String from the Program area to the String Space
>4D04	>4DB0 Second entry point for executing a Basic Program
>4D06	>56BB Subroutine to find line number after BREAK
>4D08	>5613 Subroutine that sets the pointer for next Data item
>4D0A	>5645 Subroutine to convert line number into ASCII (Trace mode)
>4D0C	>4DBF Third entry point for executing a Basic program (CONT)
>4D0E	>4E38 Subroutine that BREAKs a running program
>4D10	>4D8A First entry point for executing a Basic program (RUN)
>4D12	>515C Subroutine that sets up room for a String
>4D14	>55BB Subroutine that clears out a temporary String
>4D16	>56E1 Subroutine to convert a String into a Number
>4D18	>51A9 Garbage Collection subroutine.
>4D1A	**SUBPROGRAM POINTER TABLE** (For CALL xxxx...)
>4D1A	>4D24 Points to next Subprogram
>4D1C	>3538 Entry point for this Subprogram
>4D1E	>05 Length of this name
>4D1F	>534F554E44 :**SOUND**:
>4D24	>4D2E Points to next Subprogram
>4D26	>351C Entry point for this Subprogram
>4D28	>05 Length of this name
>4D29	>434C454152 :**CLEAR**:
>4D2E	>4D38 Points to next Subprogram
>4D30	>5713 Entry point for this Subprogram
>4D32	>05 Length of this name
>4D33	>434F4C4F52 :**COLOR**:
>4D38	>4D42 Points to next Subprogram
>4D3A	>56EF Entry point for this Subprogram
>4D3C	>05 Length of this name
>4D3D	>4743484152 :**GCHAR**:

	SUBPROGRAM POINTER TABLE Cont.
>4D42	>4D4C Points to next Subprogram
>4D44	>360E Entry point for this Subprogram
>4D46	>05 Length of this name
>4D47	>4843484152 :**HCHAR:**
>4D4C	>4D56 Points to next Subprogram
>4D4E	>362A Entry point for this Subprogram
>4D50	>05 Length of this name
>4D51	>5643484152 :**VCHAR:**
>4D56	>4D5F Points to next Subprogram
>4D58	>3643 Entry point for this Subprogram
>4D5A	>04 Length of this name
>4D5B	>43484152 :**CHAR:**
>4D5F	>4D67 Points to next Subprogram
>4D61	>3708 Entry point for this Subprogram
>4D63	>03 Length of this name
>4D64	>4B4559 :**KEY:**
>4D67	>4D71 Points to next Subprogram
>4D69	>3748 Entry point for this Subprogram
>4D6B	>05 Length of this name
>4D6C	>4A4F595354 :**JOYST:**
>4D71	>0000 Points to next Subprogram (no more)
>4D73	>37BF Entry point for this Subprogram
>4D75	>06 Length of this name
>4D76	>53435245454E :**SCREEN:**
>4D7C	Generate 'BAD VALUE' Error Message
>4D81	Generate 'STRING-NUMBER MISMATCH' Error Message
>4D86	>56D4 - Branch to routine that sets up format for screen
>4D88	>566C - Branch to CAN'T DO THAT Error
>4D8A	**RUN** - This is where a Basic program first starts to RUN. This sets up the line number pointers, scrolls the screen up 1 line and falls through to the next entry.
>4DB0	**EXECUTE** - This starts execution of the program or if in Command mode it executes the statement you just typed in.
>4DBF	Third Entry point for Basic program execution. This is where the CONTINUE Command branches to.
>4E38	Subroutine that BREAKs a running program. It prevents a break while GROM is executing, sets up the BREAK message and displays the line number.
>4E5B	**** DONE **** - This is the normal end of program subroutine.
	VECTOR TABLE FOR BASIC RESERVED WORDS
>4E84	>4FB6 FOR
>4E86	>5463 BREAK
>4E88	>5479 UNBREAK
>4E8A	>5459 TRACE
>4E8C	>545E UNTRACE
>4E8E	>400E READ
>4E90	>4004 PRINT
>4E92	>50DB CALL
>4E94	>5111 QUOTED STRING CONSTANT
>4E96	>400C RESTORE
>4E98	>50CB RANDOMIZE
>4E9A	>4006 INPUT
>4E9C	>4008 OPEN
>4E9E	>400A CLOSE
>4EA0	>4F99 ((Left Parenthesis)

	VECTOR TABLE FOR BASIC RESERVED WORDS Cont.
>4EA2	>4FB2 + (Plus)
>4EA4	>4FA8 - (Minus)
>4EA6	>4ED1 ABS
>4EA8	>4EDC ATN
>4EAA	>4EE2 COS
>4EAC	>4EE8 EXP
>4EAE	>4EEE INT
>4EB0	>4EFA LOG
>4EB2	>4F26 SGN
>4EB4	>4F40 SIN
>4EB6	>4F46 SQR
>4EB8	>4F4C TAN
>4EBA	>52BE LEN
>4EBC	>53EA CHR$
>4EBE	>4F00 RND
>4EC0	>4000 DISPLAY
>4EC2	>4002 DELETE
>4EC4	>524A SEG$
>4EC6	>531A STR$
>4EC8	>5349 VAL
>4ECA	>53A9 POS
>4ECC	>5306 ASC
>4ECE	>05 401C EOF
NOTE-------	Rather than document each of the above items, which would require another 4-6 pages of memory maps, we will talk about these routines in general. First off, many of these routines end with the Opcode of >10 this is the same as Basic's CONT, so the interpreter will go back to >4DBF and grab the next statement in your Basic Program. All of these routines use various parts of Scratch Pad RAM with FAC (>834A) and ARG (835C) being used very heavily. There is also a 24 byte segment at the top of Scratch Pad RAM (>8300 through >8316) used by Basic as temporary storage places for many of its routines. Some of the routines will clear out any values it has place into the FAC and ARG area or the Row, Column and Character value area at >837D - >837F. Most of the String handling routines require that FAC through FAC + 7 (>834A - >8351) be set up prior to execution as follows: >834A = The Symbol table address that points to the string. >834C = >6500 for a string and >6400 for numerics. >834E = The address in VDP RAM of the string. >8350 = The length of the string. *>8352 - Sometimes the GROM Flag is temporarily stored here
>54CF	Subroutine to handle User Defined Functions (ie: DEF)
>5600	Subroutine to check for String or numeric and set register bits.
>5613	Subroutine to set the pointer for DATA items.
>5645	Subroutine to convert the Line number into ASCII.
>565C	Subroutine to print out an Error Message.
>56BB	Subroutine to find line # after BREAK, UNBREAK or RESTORE.
>56EF	**GCHAR** subroutine.
>5713	**COLOR** subroutine.
>5740	Subroutine to convert floating point to integer.
>5755	Subroutines used by CALL JOYST and CALL KEY.
>57AB	Subroutine to check for the left parenthesis (.
>57C0	Error Message subroutines.

CONSOLE CRU BIT MAP (9901)

```
+------------------------------------------------------------------------+
|                      | All of the Data for the 9901 on the 99/4A is inverted. |
|----------------------| On or Set = 0    and    Off or Reset = 1              |
|Cru Base| Bit |-------------------------------------------------------------|
|Address | No. | Description                                                 |
|--------|-----|-------------------------------------------------------------|
|>0000   |  0  | 0 = Internal 9901 Control        1 = Clock Control          |
|>0002   |  1  | Set by an External Interrupt (Peripheral Device)            |
|>0004   |  2  | Set by TMS 9918A on Vertical Retrace Interrupt              |
|>0006   |  3  | Set by Clock Interrupt for Cassette read/write routines     |
|        |     |      Also used for Keyboard Matrix Row 7                    |
|        |     |                                                             |
|        |     |                KEYBOARD 8 x 8 MATRIX                        |
|        |     |   Column   0      1   2   3   4   5    6    7               |
|        |     |         +-------------------------------------------+       |
|>0006   |  3  | Row 7   |   =      .   ,   M   N   /   Joy1 Joy2 Fire  |    |
|>0008   |  4  | Row 6   | SPACE    L   K   J   H   ;   Joy1 Joy2 Left  |    |
|>000A   |  5  | Row 5   | ENTER    O   I   U   Y   P   Joy1 Joy2 Right |    |
|>000C   |  6  | Row 4   |          9   8   7   6   0   Joy1 Joy2 Down  |    |
|>000E   |  7  | Row 3   | FCTN     2   3   4   5   1   Joy1 Joy2 Up    |    |
|>0010   |  8  | Row 2   | SHIFT    S   D   F   G   A                   |    |
|>0012   |  9  | Row 1   | CTRL     W   E   R   T   Q                   |    |
|>0014   | 10  | Row 0   |          X   C   V   B   Z                   |    |
|        |     |         +-------------------------------------------+       |
|        |     |                                                             |
|>0016   | 11  | Not Used                                                    |
|>0018   | 12  | Reserved - High Level                                       |
|>001A   | 13  | Not Used                                                    |
|>001C   | 14  | Not Used                                                    |
|>001E   | 15  | Not Used                                                    |
|        |     |                                                             |
|>0020   | 16  | Reserved                                                    |
|>0022   | 17  | Reserved                                                    |
|>0024   | 18  | Bit 2 of Keyboard Matrix Column select (8x8 matrix)         |
|>0026   | 19  | Bit 1 of Keyboard Matrix Column select                      |
|>0028   | 20  | Bit 0 of Keyboard Matrix Column select (MSB)                |
|        |     |  ( set up the column to read -  R1 = 00xx thru 07xx    )   |
|        |     |  (      LI R12,>0024            LDCR R1,3              )   |
|        |     |  ( and read the row bits (3-10) with                   )   |
|        |     |  (      LI R12,>0006            STCR R4,8   INV R4     )   |
|        |     |                                                             |
|>002A   | 21  | Set Alpha Lock                                              |
|>002C   | 22  | Cassette CS1 motor control  On/Off                          |
|>002E   | 23  | Cassette CS2 motor control  On/Off                          |
|>0030   | 24  | Audio Gate enable/disable                                   |
|>0032   | 25  | Cassette Tape Out                                           |
|>0034   | 26  | Reserved                                                    |
|>0036   | 27  | Cassette Tape In                                            |
|>0038   | 28  | *                                                           |
|        |     | * Not Used - causes lock up.                                |
|>00FE   |128  | *                                                           |
+------------------------------------------------------------------------+
```

9900 MICROPROCESSOR INSTRUCTIONS

Inst	Status Bits L>	A>	EQ	C	OV	OP	X	Result compared to zero	Description
A	x	x	x	x	x	.	.	yes	Add words
AB	x	x	x	x	x	x	.	yes	Add Bytes
ABS	x	x	x	x	x	.	.	no	Absolute Value
AI	x	x	x	x	x	.	.	yes	Add word with Immediate value
ANDI	x	x	x	.	.	.	.	yes	AND word with Immediate value
B	.	.	.	.	.	.	.	no	Branch (Goto)
BL	.	.	.	.	.	.	.	no	Branch & Link (Gosub - R11=Return Addr)
BLWP	.	.	.	.	.	.	.	no	Branch & Load Workspace Pointer
C	x	x	x	.	.	.	.	no	Compare words
CB	x	x	x	.	.	x	.	no	Compare Bytes
CI	x	x	x	.	.	.	.	yes	Compare word with Immediate value
CKOF	.	.	.	.	.	.	.	no	External Clock Off - not on 4A
CKON	.	.	.	.	.	.	.	no	External Clock On - not on 4A
CLR	.	.	.	.	.	.	.	no	Clear (make it >0000)
COC	.	.	x	.	.	.	.	no	Compare Ones Corresponding
CZC	.	.	x	.	.	.	.	no	Compare Zeros Corresponding
DEC	x	x	x	x	x	.	.	yes	Decrement
DECT	x	x	x	x	x	.	.	yes	Decrement by Two
DIV	.	.	.	.	x	.	.	no	Divide (unsigned)
IDLE	.	.	.	.	.	.	.	no	Idle - Wait for Interrupt - not on 4A
INC	x	x	x	x	x	.	.	yes	Increment
INCT	x	x	x	x	x	.	.	yes	Increment by Two
INV	x	x	x	.	.	.	.	yes	Invert (same as NOT)
JEQ	.	.	1	.	.	.	.	no	Jump if Equal (or Zero)(EQ=1)
JGT	.	1	.	.	.	.	.	no	Jump if Greater Than (A>=1)
JH	1	.	0	.	.	.	.	no	Jump if High (L>=1 and EQ=0)
JHE	1	.	1	.	.	.	.	no	Jump if High or Equal (L>=1 or EQ=1)
JL	0	.	0	.	.	.	.	no	Jump if Low (L>=0 and EQ=0)
JLE	0	.	1	.	.	.	.	no	Jump if Low or Equal (L>=0 or EQ=1)
JLT	.	0	0	.	.	.	.	no	Jump if Less Than (A>=0 and EQ=0)
JMP	.	.	.	.	.	.	.	no	Jump - always (unconditional)
JNC	.	.	.	0	.	.	.	no	Jump if No Carry (C =0)
JNE	.	.	0	.	.	.	.	no	Jump if Not Equal (EQ=0)
JNO	.	.	.	.	0	.	.	no	Jump if No Overflow (OV=0)
JOC	.	.	.	1	.	.	.	no	Jump On Carry (C =1)
JOP	.	.	.	.	.	1	.	no	Jump if Odd Parity (OP=1)
LDCR	x	x	x	x	.	b	.	yes	Load Cru Bits (Write Out Bits)
LI	x	x	x	.	.	.	.	yes	Load with Immediate value
LIMI	.	.	.	.	.	.	.	no	Load Interrupt Mask Immediate
LREX	.	.	.	.	.	.	.	no	Load External - not on 4A
LWPI	.	.	.	.	.	.	.	no	Load Workspace Pointer Immediate

9900 MICROPROCESSOR INSTRUCTIONS

Inst	L>	A>	EQ	C	OV	OP	X	Result compared to zero	Description
MOV	x	x	x	.	.	.	.	yes	Move word
MOVB	x	x	x	.	.	x	.	yes	Move Byte
MPY	.	.	.	.	.	.	.	no	Multiply (unsigned)
NEG	x	x	x	x	x	.	.	yes	Negate (same as Change Sign or NOT+1)
NOP	.	.	.	.	.	.	.	no	No Operation - Pseudo (JMP $+2)
ORI	x	x	x	.	.	.	.	yes	OR with Immediate value
RSET	.	.	.	.	.	.	.	no	External Reset - not on 4A
RT	.	.	.	.	.	.	.	no	Return - Pseudo (B *R11)
RTWP	x	x	x	x	x	x	x	no	Return with Workspace Pointer
S	x	x	x	x	x	.	.	yes	Subtract words
SB	x	x	x	x	x	x	.	yes	Subtract Bytes
SBO	.	.	.	.	.	.	.	no	Set Bit to One
SBZ	.	.	.	.	.	.	.	no	Set Bit to Zero
SETO	.	.	.	.	.	.	.	no	Set to Ones (make it >FFFF)
SLA	x	x	x	x	x	.	.	yes	Shift Left Arithmetic
SOC	x	x	x	.	.	.	.	yes	Set Ones Corresponding
SOCB	x	x	x	.	.	x	.	yes	Set Ones Corresponding Bytes
SRA	x	x	x	x	.	.	.	yes	Shift Right Arithmetic
SRC	x	x	x	x	.	.	.	yes	Shift Right Circular
SRL	x	x	x	x	.	.	.	yes	Shift Right Logical
STCR	x	x	x	.	.	b	.	yes	Store Cru Bits (Read In Bits)
STST	.	.	.	.	.	.	.	no	Store Status Register
STWP	.	.	.	.	.	.	.	no	Store Workspace Pointer
SWPB	.	.	.	.	.	.	.	no	Swap Bytes
SZC	x	x	x	.	.	.	.	yes	Set Zeros Corresponding
SZCB	x	x	x	.	.	x	.	yes	Set Zeros Corresponding Bytes
TB	.	.	x	.	.	.	.	no	Test Bit
X	e	e	e	e	e	e	e	yes	Execute
XOP	e	e	e	e	e	e	x	no	Extended Operation - Software Interrupt
XOR	x	x	x	.	.	.	.	yes	XOR - Exclusive OR

b - Odd Parity bit is only affected on byte type Load Cru and Store Cru instructions (8 bits or less).

e - The Execute instruction does not affect status bits, but the instruction that the Execute instruction executes may. The XOP instruction sets the X status bit and the instruction that the XOP branches to may affect the status bits.

NOTE: If the Result is compared to zero it will set the E (Equal) when it is zero and clear the E bit in the Cpu Status Register when it is not zero. So if the instruction is MOV R0,R0 and R0 contains zero, the E bit will be set and the next instruction may be JNE >xxxx (Jump if Not Equal) which, in this case, has the same meaning as a Jump If Not Zero instruction would.

BREAK POINT WORK SHEET

CPU BP SETTINGS

Key Scan Routine E/A and GPL	02B2	______
Key Press Detected & Decoded	0444	______
Read Key Board Cru Bits	0346	______
Start Execution Of Interrupt Routine	0900	______
______________		______
______________		______
______________		______

GROM BP SETTINGS

Reset V1 to Watch Power Up	00EB	______
Power Up Title Screen Built	6000	______
______________		______
______________		______
______________		______
______________		______

VDP BP SETTINGS

Start Screen Scroll	0020	______
End Screen Scroll (32 Column)	4300	______
Start Write To Basic Crunch Buffer	4320	______
Start Write To X-B Crunch Buffer	4820	______
Reset Screen Color in Basics	8707	______
______________		______
______________		______
______________		______

GROM START ADDRESS & VDP REGISTER

WORK SHEET

POWER UP ROUTINE

```
Cpu                   Grom
WS 83E0 ________      AD xxxx ________
PC 0024 ________

  v0 00   v2 F0   v4 F9   v6 F8
  v1 E0   v3 0E   v5 86   v7 F7
```

TI BASIC

```
Cpu                   Grom
WS 83E0 ________      AD 216F ________
PC 006A ________

  v0 00   v2 F0   v4 F9   v6 F8
  v1 E0   v3 0E   v5 86   v7 F7
```

TI EXTENDED BASIC

```
Cpu                   Grom
WS 83E0 ________      AD 6372 ________
PC 006A ________

  v0 00   v2 00   v4 00   v6 00
  v1 E0   v3 20   v5 06   v7 07
```

EDITOR/ASSEMBLER

```
Cpu                   Grom
WS 83E0 ________      AD 6025 260
PC 006A ________

  v0 00   v2 00   v4 01   v6 00
  v1 E0   v3 0E   v5 06   v7 F5
```

MINI MEMORY

```
Cpu                   Grom
WS 83E0 ________      AD 6020 ________
PC 006A ________

  v0 00   v2 00   v4 01   v6 00
  v1 E0   v3 0E   v5 06   v7 F5
```

EASY BUG

```
Cpu                   Grom
WS 83E0 ________      AD 70B9 ________
PC 006A ________

  v0 00   v2 F0   v4 F9   v6 F8
  v1 E0   v3 0E   v5 86   v7 F7
```

TERMINAL EMULATOR II - Title & Menu

```
Cpu                   Grom
WS 83E0 ________      AD 6292 ________
PC 006A ________

  v0 00   v2 00   v4 01   v6 00
  v1 E0   v3 0F   v5 08   v7 CF
```

ADVENTURE

```
Cpu                   Grom
WS 83E0 ________      AD 6798 ________
PC 006A ________

  v0 00   v2 00   v4 01   v6 F8
  v1 F0   v3 0F   v5 86   v7 4B
```

SPEECH EDITOR

```
Cpu                   Grom
WS 83E0 ________      AD 6075 ________
PC 006A ________

  v0 00   v2 F0   v4 F9   v6 F8
  v1 E0   v3 0E   v5 86   v7 F7
```

MUNCH MAN - Title

```
Cpu                   Grom
WS 83E0 ________      AD 6020 ________
PC 006A ________

  v0 00   v2 00   v4 01   v6 00
  v1 E0   v3 0E   v5 06   v7 03
```

GROM START ADDRESS & VDP REGISTER

WORK SHEET

PARSEC - Title

Cpu		Grom	
WS 83E0 ________		AD 601D ________	
PC 006A ________			

v0 00	v2 00	v4 01	v6 00
v1 E2	v3 0E	v5 06	v7 11

PARSEC - Game

Cpu		Grom	
WS 83E0 ________		AD 60B7 ________	
PC 006A ________			

v0 02	v2 06	v4 03	v6 03
v1 E2	v3 FF	v5 36	v7 00

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

Cpu   Grom
WS ______________ AD ______________
PC ______________

v0 ____ v2 ____ v4 ____ v6 ____
v1 ____ v3 ____ v5 ____ v7 ____

MILLERS GRAPHICS - LIMITED WARRANTY

Millers Graphics warrants the Explorer program, which it manufactures, to be free from defects in materials and workmanship for a period of 90 days from the date of purchase.

During the 90 day warranty period Millers Graphics will replace any defective products at no additional charge, provided the product is returned, shipping prepaid to Millers Graphics. The Purchaser is responsible for insuring any product so returned and assumes the risk of loss during shipping.

Ship to:

Millers Graphics
1475 W. Cypress Ave.
San Dimas, California 91773

WARRANTY COVERAGE

This EXPLORER program is warranted against defective material and workmanship. THIS WARRANTY IS VOID IF THE PRODUCT HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, TAMPERING, IMPROPER SERVICE OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR WORKMANSHIP.

WARRANTY DISCLAIMERS

ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE 90 DAY PERIOD. MILLERS GRAPHICS. SHALL NOT BE LIABLE FOR LOSS OR USE OF THE SOFTWARE OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USE.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusion may not apply to you in those states.

LEGAL REMEDIES

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

REPLACEMENT AFTER WARRANTY

After the 90 Warranty period has expired you may return any original defective diskette, along with a check for 4.00 to cover shipping and diskette costs, and we will replace it.

MILLERS GRAPHICS
1475 W. Cypress Ave.
San Dimas, CA 91773