SUPERBUG II

Version 2.0

by

Edgar L. Dohmann
Rt. 5 Box 84
Alvin, Texas 77511

```
******                                                      *******
*******  SUPERBUG II    VERSION 2.0    1-MAY-1986          ********
********                                                  *********
*********              BY: Edgar L. Dohmann              **********
**********             (C) COPYRIGHT 1986               ***********
```

This debugger program was derived from the SUPERBUG program which Texas Instruments released to public domain. SUPERBUG II includes fixes to several bugs in the TI version, enhancements to many of the original features, and the addition of some new features.

SUPERBUG II is being distributed as a FAIRWARE product by its author. You are welcome to distribute copies of the SUPERBUG II disk to anyone who wants one. The manual however is copyrighted and authorized copies of the manual can only be obtained from the author. Anyone who receives a FAIRWARE copy of the disk may receive a printed copy of this manual by sending $5.00 to the author at the address below.

Since the disk includes several files for the various loading options of SUPERBUG II, the author specifically requests that none of the SUPERBUG II files be placed on any BBS system for downloading. The purpose of this policy is to ensure that everyone will receive all of the files on the distribution disk. Therefore, the only authorized means for distributing SUPERBUG II is by disk copy.

This release of SUPERBUG II includes several new features plus a number of enhancements to features which were in Version 1.0. Substantial code optimization was necessary in order to provide these features and still keep the size under 8 Kbytes. I would like to thank Ewell Brigham of Baytown, Texas for his assistance in preparing this new version of SUPERBUG II.

SUPERBUG II is available through FAIRWARE or directly from the author. Anyone who receives a FAIRWARE copy is asked to send $5.00 and in return they will receive a printed copy of this manual. Copies of the disk and manual may also be ordered from the author for $10.00 at the following address:

EDGAR L. DOHMANN
RT. 5 BOX 84
ALVIN, TEXAS 77511

Because of the extensive updates to the program and manual and considering the minimal price requested for the program, I cannot afford to offer any other upgrade service to those who have already purchased Version 1.0. I really do appreciate all orders and contributions that I have already received. Your comments, suggestions, and support have provided the incentive for this new version. Thank you and good luck with SUPERBUG II!

CONTENTS

PAGE

INTRODUCTION

SUPERBUG II is a stand alone program that may be loaded by the Editor/Assembler LOAD AND RUN option or the CALL LOAD option of TI-BASIC or EXTENDED BASIC. MEMORY EXPANSION and a DISK DRIVE SYSTEM are required. The RS232 or DISK may optionally be used to get a hard copy printout or file dump on some operations.

SUPERBUG II is a very powerful debug tool which provides many functions usually only available on very expensive development systems requiring special hardware. SUPERBUG II allows you to actually step through your machine language programs in ROM or RAM, executing each machine instruction one at a time. This allows you to examine the logic of your program as it is being run. As each instruction is executed, the SYMBOLIC interpretaion is displayed on the screen in the same format as it occurs in your assembley source listing, providing a trace of instruction execution.

SUPERBUG II has a built in DISS-ASSEMBLER which you can use to decipher machine code to its symbolic assembly language representation. It will interpret any instruction and show all types of operand uses. It will even display the JMP address of jump instructions. A new feature allows disassembly of obvious TEXT or DATA areas of a program as DATA statements only. This will bypass the DIS-ASSEMBLER's attempt to translate these areas into mnemonics.

Operation of SUPERBUG II is syntactally identical to the TI-DEBUGGER program, however the A, I, and V features provided by the TI program are not included in SUPERBUG II due to memory size limitations and because the author did not feel they were particularly useful. In addition, SUPERBUG II has several features that were not in the TI DEBUGGER or the original SUPER-BUGGER.

SUPERBUG II is a complete update of SUPER-BUGGER which was released to public domain by Texas Instruments. The version released by TI had several bugs and lacked some of the features of TI-DEBUGGER to keep its size under 7 Kbytes. In making SUPERBUG II, an attempt was made to fix all known bugs, add some additional useful features, and keep the total size of the program under 8 Kbytes. A version has also been created which has an AORG at >6000 for those of you with SUPER SPACE or SUPER CART cartridges, a Morning Star 128K RAM card, a GRAM Kracker, or other means of loading a program into the cartridge memory space from >6000 to >7FFF.

VERSION 1.0 BUG FIXES

The following bugs in the TI release were corrected in Version 1.0 of SUPERBUG II:

1) Memory Dumps (D) or Disassemblies (A) may be directed to a Disk or to a printer.

2) Memory Dumps and Disassemblies to >FFFE or >FFFF will not rollover to >0000 and keep going if >FFFE or >FFFF is specified as the stop address.

3) Some minor errors in the Disassembler have been fixed. MPY and DIV formats were not exactly right and the Shift & CRU formats did not use a > sign for operands from >A to >F.

5) The program was not re-entrant because some of the data buffers overlaid initialization code. This was usually not a problem if the program was not restarted after exiting except by reloading. However one of the goals of SUPERBUG II was to create a version for loading at >6000 with a GROM header so it could be restarted from the startup menu for those users with hardware that supports RAM in the cartridge memory space.

6) The original version was difficult to use in an Extended Basic environment. The version supplied for use with Extended Basic was in standard object format which (due to X-Basic's inefficient loader) took a long time to load. This program loaded into Low Memory and took up almost all of the available space so there was no room left to load any routines to test with SUPER-BUGGER. To top things off, the sketchy documentation did not explain how to start the program in an X-Basic environment.

   SUPERBUG II includes a small loader in standard object format which only occupies 618 bytes of Low Memory leaving over 6 Kbytes of Low Memory for loading routines to be tested. The small loader then loads a SUPERBUG II program file into High Memory starting at >A000. The total time for loading the SUPERBUG II loader and program are much faster than the old large oject file.

VERSION 2.0 BUG FIXES

The following bugs remaining in Version 1.0 have been corrected in Version 2.0 of SUPERBUG II:

1) Memory Dumps and Disassemblies which cross address >8000 now work properly.

2) The operand value for jump instructions is now computed properly in the Disassembler. The leading zero is also removed from registers R0 through R9 for instructions that generate register references. These two changes make it possible to reassemble source files generated by the Disassembler without having to edit these types of instructions.

3) Memory Dumps from GROM or VDP to an output device now print the right addresses.

4) The output device is now closed only when dumps or disassemblies are actually directed to the output device. This corrects a bug that only showed up when the SUPER SPACE version was restarted after a dump or disassembly was directed to the screen.

5) The Q and E commands now work properly in all cases.

6) Individual breakpoints can now be deleted if the breakpoint buffer is full.

7) Some typographical errors in the manual have been corrected. The version 1.0 manual incorrectly stated that 16 breakpoints can be set at once. The correct number is 11. The program exit was described as SHIFT/Q in the Version 1.0 manual. SHIFT/Q is correct for a TI-99/4 but FCTN/= must be used with a TI-99/4A.

8) The original version was difficult to use in a Console Basic environment. A new entry point has been added so the startup prompts will be visible and VDP register control has been added for compatibility with the Console Basic environment.

VERSION 1.0 ENHANCEMENTS

The following enhancements were added to SUPERBUG II when Version 1.0 was released:

1) An O option has been added to allow the List device to be changed without having to find it in RAM then edit it with the M option. This feature is especially handy when a series of dumps are to be made to various disk files.

2) The C, F, H, K, N, >, and . commands from TI DEBUGGER have been restored to SUPERBUG II. The C and N commands are handy for debugging or examining DSR ROMs.

3) A (-) termination has been added to the A and D commands. Previously, when a disassembly or dump went to printer, the Address was included but not when output was sent to disk. Now they can be included to either disk or printer if the command is terminated with an ENTER but can be eliminated from both if the command is terminated with a MINUS sign.

4) A (:) and (;) termination have been added to the A command to force disassembly to DATA statements. This bypasses the attempt of the disassembler to decode TEXT and DATA areas into mnemonics. All DATA statements include an ASCII translation of the bytes as a comment. The (:) termination works like an ENTER and the (;) termination works like the (-) termination for printer and disk dumps.

5) A (:) and (;) termination have also been added to the N command. These terminations will cause the block transfer to be executed in slow mode (10 to 20 milliseconds per byte) so you can transfer a block of memory into EEPROM devices. The (:) termination works like the carriage return termination while the (;) termination works like the (-) termination for normal speed transfers.

6) A J option has been added which allows you to toggle the screen colors through a choice of 5 different foreground and background combinations.

7) The list device output specification has been changed from OUTPUT to APPEND. This has no effect on printer output but it allows you to direct a series of A or D outputs to a disk file without having to open a new file each time.

8) A relocation option has been added to the A and D commands so when you dump or disassemble blocks of memory that have been relocated from another location, the addresses on the dump will reflect the original location of the data.

9) Some code tightening has been done to partially offset the additional space required for the new features.

VERSION 2.0 ENHANCEMENTS

The following enhancements have been added to SUPERBUG II in Version 2.0:

1) The CTRL/L and CTRL/S commands have been added to allow assembly language PROGRAM files to be loaded and saved. The Load command is sometimes better for debugging purposes than Option 5 of Editor/Assembler because control is returned to SUPERBUG II rather than to the program that was loaded. The Save feature is more convenient than the SAVE utility on the Editor/Assembler disk because the labels SFIRST, SLAST, and SLOAD are not required.

2) The output device/file pathname is expanded from 20 to 28 characters. This provides better functionality for those users with hard disk systems who can enter file pathnames which are longer than 15 characters. The file pathname for loading and saving can also be 28 characters long.

3) The I, CTRL/I, and CTRL/F commands have been added to allow a 10-character ASCII string to be defined, inserted into memory, and searched for.

4) The G command to change GROM base has been added. This feature from the original TI Debugger has been omitted from previous releases because no commercial hardware existed which would support any GROM base addresses other than the default. New hardware is under development by several manufacturers that should support the REVIEW MODULE LIBRARY function of the TI-99/4A operating system. This new command should allow SUPERBUG II to be compatible with this new hardware whenever it becomes available.

5) The M command has been modified so that GRAM writes will be compatible with GRAM Kracker. This modification will make SUPERBUG II incompatible with any GRAM emulators that require an address decrement before writing to GRAM. Since TI never made such emulators commercially available, this should not be a serious limitation.

6) The 5X7 character sets are loaded automatically when the the SUPER SPACE version is executed. This is much more readable than the 6X8 character set.

7) Special characters are defined for the CTRL/F, CTRL/I, CTRL/L, and CTRL/S commands. This allows a unique character to be displayed as the echo for these commands.

8) A 6th color option (black on white) is added to the J command. This is a popular choice for users with monochrome monitors. The border is also set to match the background color in the J command.

9) Error messages are now displayed if an error occurs during a file open or close.

10) The current output device is now displayed when the O command is used. The same message format is also used with the I, CTRL/L, and CTRL/S commands.

11) A significant amount of code optimization was required to make room for the new features and improvements of Version 2.0. This optimization was intended mainly to conserve memory space but in some cases the program performance was also improved.

COMPARISON TO TI-DEBUGGER

The following is a summary comparison of the TI DEBUGGER and SUPERBUG II. If you have the TI SUPER-BUGGER, you will note that the O, J, I, CTRL/F, CTRL/I, CTRL/L, and CTRL/S functions are new and the G, H, >, ., N, F, K, and P functions from the TI DEBUGGER have been restored. Essentially only the A, I, and V functions from the original TI DEBUGGER are still omitted, primarily because of their lack of value. The size of SUPERBUG II is less than 8 Kbytes, but is about 1 Kbyte larger than the original SUPER-BUGGER.

COMMAND	SUPERBUG II	TI-DEBUGGER
A	DIS-ASSEMBLE machine code	Load Memory with ASCII
B	Breakpoint Set/Clear (2-wd)	Same except 1-wd on some 4As
C	CRU Inspect/Change	Same
D	Dump Memory to List Device	**Not Supported
E	Execute	Same
F	Find Word or Byte	Same
CTRL/F	Find String	**Not Supported
G	GROM Base Change	Same
H	Hex Arithmetic	Same
I	Enter String	Inspect Screen Location
CTRL/I	Insert String	**Not Supported
J	Toggle Screen Colors	**Not Supported
K	Find Data Not Equal	Same
L	Toggle List Device on/off	**Not Supported
CTRL/L	Load PROGRAM File	**Not Supported
M	Memory Inspect/Change	Same
N	Move Block	Same but no slow option
O	Change Output Device Name	**Not Supported
P	Compare Memory Block	Same
Q	QIUT Debugger	Same
R	Inspect/Change WP,PC,SR	Same
S	Single Step on any TI99/4A	Step with special Hrdw.
CTRL/S	Save PROGRAM File	**Not Supported
T	Trade User/SBUG Screen	Trade screens
U	Toggle Basic offset on/off	Same
V	Run till VALUE = entered #	VDP Base Change
W	Inspect/Change Register	Same
X,Y,or Z	Change BIAS	Same
>	Hex to Decimal convert	Same
.	Decimal to Hex convert	Same

LOADING SUPERBUG II

The SUPERBUG II distribution disk contains the following 3 files which provide various methods of loading the program into memory:

1) SBUGO -- This is a tagged object file in condensed format. It can only be loaded with a loader like those in the Editor/Assembler and Mini Memory cartridges. The Extended Basic cartridge will not load this version. This version is relocatable so the loader can load it into any 8K block that is available to it.

2) SBUG -- This is a program file that can be loaded with the Editor/Assembler or with Extended Basic using the companion file LOADSBUG. This version will load from >A000 to just under >BFFF. This forces SUPERBUG II into High Memory so you can debug your Extended Basic compatible assembly routines which reside in Low Memory.

3) SBUG6 -- This is a program file that can be loaded with Editor/Assembler. This version loads from >6000 to just under >7FFF. To use this file you will need a Morning Star 128K card, a SUPER SPACE or SUPER CART cartridge, or some other means of loading into cartridge memory space. This version contains a GROM header so on most consoles, subsequent resets will allow selection of the program from the startup menu.

LOADING WITH MINI MEMORY

Select   1 LOAD AND RUN

When this prompt appears   * LOAD AND RUN *
                             FILE NAME

Type in   DSK1.SBUGO   and press ENTER

The Mini-Memory loader will then load the program and return to the FILE NAME prompt when it is through. If you have not yet loaded your program to test, then type the name of your program. After you have loaded all object files you wish, then merely press enter to advance to the PROGRAM NAME prompt. At this time type SBUG and SUPERBUG II will start running.

LOADING WITH EDITOR/ASSEMBLER

Select   3 LOAD AND RUN

When this prompt appears   * LOAD AND RUN *
                             FILE NAME

Type in   DSK1.SBUGO   and press ENTER

The Editor/Assembler loader will then load the program and return to the FILE NAME prompt when it is through. If you have not yet loaded your program to test, then type the name of your program. After you have loaded all object files you wish, then merely press enter to advance to the PROGRAM NAME prompt. At this time type SBUG and SUPERBUG II will start running.

Alternatively, with Editor/Assembler, you can load SUPERBUG II in the following manner:

Select   5 RUN PROGRAM FILE

When this prompt appears   * RUN PROGRAM FILE *
                             FILE NAME

Type in   DSK1.SBUG
     or   DSK1.SBUG6   and press ENTER

The Editor/Assembler program loader will then load and run SUPERBUG II at >6000 or >A000 depending on which version you loaded. It is recommended that you already have your program to debug loaded into memory before loading the SBUG or SBUG6 program files.

If you loaded the SBUG6 version and you have a WIDGET or other device with a RESET pushbutton, you can restart SUPERBUG II by making the appropriate selection from the startup menu. If you use the cartridge RAM version of SUPERBUG II, you should either have battery backed up RAM in the >6000 to >7FFF range or a RESET pushbutton if your memory is not battery backed up. If you have to use the power switch to reset your computer and your memory is not battery backed up, you will likely lose the contents of your cartridge RAM.

LOADING WITH EXTENDED BASIC

Before loading SUPERBUG II with Extended Basic you will first have to execute the CALL INIT command either from the Command/Edit Mode or from within a program. After the CALL INIT is completed, execute the command CALL LOAD("DSK1.LOADSBUG"). This loads a small loader program which is more efficient than the standard Extended Basic loader. After loading this special loader program and any routines you want to test, enter the command CALL LINK("GLOAD").

These CALLs can be issued from command mode or from within a program, however if you issued them from a program, the X-Basic program will not be intact once SUPERBUG II is loaded by the CALL LINK("GLOAD") command. SUPERBUG II will load into the High Memory space from >A000 to just under >BFFF.

The LOADSBUG program is 618 bytes long which still leaves over 6 Kbytes of space for your assembly routines in Low Memory. However, If LOADSBUG will not fit into Low Memory with your routines, you may have to break your routines into smaller blocks of code for testing.

While DSK1 is used in all the examples here, any drive containing the SUPERBUG II programs can be used for all except the Extended Basic loaders. Disk 1 must be used when loading under Extended Basic because when you issue the CALL LINK("GLOAD)" command to run LOADSBUG, it will expect to find the SBUG file on DSK1.

LOADING WITH CONSOLE BASIC

To load SUPERBUG II from Console Basic, you must have an Editor/Assembler or Mini Memory cartridge plugged into your console. Alternatively, a cartridge like SUPER SPACE can be used because it contains an Editor/Assembler GROM.

From command mode or from within a Basic program, enter the command CALL LOAD("DSK1.SBUGO"). It is not necessary to use the CALL INIT as is required with Extended Basic.

After loading the program and any routines you might want to test, enter the command CALL LINK("SBUGB"). This is a special entry point for Console Basic so the startup prompts will be visible.

STARTING SUPERBUG II

When SUPERBUG II is started, it will clear the screen and display a startup message indicating the version number of the program and an author's credit. If you are not running under the Extended Basic or Console Basic environment, you will then be prompted for an opportunity to use a BIT MAP screen.

In this case the following prompt will appear:

```
        USE BITMAP SCREEN (Y/N) ?
```

Press <N> if you are using standard TI screens. This will cause SUPERBUG II to share the screen area of VDP RAM with the user's program screen.

Press <Y> to set up VDP screen addresses which allow the user screen to be automatically restored to BIT MAP mode upon exit from SUPERBUG II into the user program. This is extremely helpful when developing software that uses different screen modes than the standard TI system screens. You may use the <T> option to flip between the user screen and the SUPERBUG screen if you have answered <Y> to this prompt.

If you are running under the Extended Basic or Console Basic environment, this prompt will not appear because BIT MAP mode is not available in these environments.

When using the BIT MAP option, SUPERBUG II will set up the VDP RAM as follows:

SUPERBUG II	USER	CONTENTS
>3800	>0000	Pattern Descripter Table
>3C00	>1C00	Screen Image Table
>3B00	>1F00	Sprite Attribute Table
>UNUSED	>1800	Sprite Description Table
>3B80	>2000	Color Table
>1000		PABs
>1026		PAB Buffer (>1026 to >127F)

The location of the SUPERBUG II PAB and PAB Buffer should prevent it from interfering with any critical data in the User Screen area in this mode, however if it does then you will not be able to dump to an output device while in BIT MAP mode.

After the BIT MAP prompt (or immediately after the version and credit messages in Extended Basic environment), you will be prompted for an output device. The following prompt will appear:

```
          Device/File:
          RS232.BA=9600
          ENTER LIST DEVICE
          *
```

Press <ENTER> to use the default output device of:

```
          RS232.BA=9600
```

If you wish to use a different output device (printer or disk), merely type the device description you wish to use after the asterisk on the blank line presented. The list device name cannot exceed 28 characters in length. If you make a mistake, merely press the FCTN and S keys for a backspace and the prompt will be repeated.

If you do not wish to use a List Device, merely press <ENTER> in response to the prompt but be sure to select the <L> command to turn off the List Device toggle so the D and A commands will be directed to the screen.

After entering a List Device, SUPERBUG II will inform you that the List Device toggle is turned on then will display a blank line with a period as the prompt for command entry.

SUPERBUG II OPERATING PROCEDURES

The syntax for SUPERBUG II is the same as the TI-DEBUGGER. The following conventions are used. Items surrounded by <angle brackets> represent mandatory data that must be entered. Items surrounded by {braces} indicate you must choose between two or more items indicated. Items surrounded by [brackets] indicate optional data. The parentheses (...) indicates the previous item may be repeated.

All SUPERBUG II commands must begin with a single character as indicated in the reference chart. Depending on the starting character, one or more additional entries must be made. Some commands (such as J, U, and L) do not even require termination with ENTER. The specific syntax for each command is described in the following sections. If additional entries are required they may be separated by spaces or commas.

With many of the commands, you can enter a G or V after an operand to indicate that the operation is to take place in GROM (Console or Command Module) or VDP RAM, respectively, instead of in CPU RAM or ROM.

You can define up to three bias characters, labeled X, Y, and Z. When one of these characters is appended to an operand, its value is added to the value of the operand. Thus if you have set X to >204 and enter an operand of >100X, the operand used is the sum of >204 and >100 or >304.

When you type the character that indicates the command you wish to execute, SUPERBUG II will automatically place a space after the character, although you cannot see it because no cursor appears on the screen. Do not press <space> after a letter unless you wish to terminate the command. While SUPERBUG II does not use a cursor, a period will appear at the beginning of a blank line when you may enter a new command.

After choosing a command, you must enter the command's operands (if any), which consist of up to three hexadecimal fields, depending on the command. Operands contain four hexadecimal digits each. If you enter more than four digits, only the last four are used. If you enter fewer than four digits, they are considered by SUPERBUG II to be the right-most digits of the operand and any missing digits will be ZERO.

If a command allows one or more operands, the command may be terminated by a space, ENTER, comma, colon, or semicolon. Depending on the command, each termination may have a different effect. If you want to escape a command once it is started, simultaneously press FCTN and X. The escape feature is particularly useful with the M, D, and A commands if you do not want to wait until the dump is complete.

A -- DISS-ASSEMBLE MACHINE CODE

* To Diss-assemble code at current User PC Location

ENTER:

```
        A [<return> or <-> or <:> or <;>]
    or A [<Number>] [<return> or <-> or <:> or <;>]
```

Diss-assembles the machine instruction/s beginning at the user PC Address. If a single paramenter is entered it is treated as the Number of instructions to display or print.

Example:

```
        A 100 <return>  Disassembles 256 (hex 100) bytes starting
                        from the current user's PC
```

* To Diss-assemble code between locations specified

ENTER:

```
        A <Start Addr> <End Addr> [<Relocation Addr>] <return>
```

Diss-assembles the machine instruction/s beginning with the Start address entered and continuing to the End address entered. If a Relocation address is specified, the disassembly will occur between the Start and End addresses, but any addresses (including JMP addresses) displayed in the listing will appear as if the Relocation address were the Start address.

Example:

```
        A C000,DFFF,4000 <->   Disassembles the code from >C000 to
                               >DFFF but any JMP addresses will
                               appear as if the code was located
                               from >4000 to >5FFF.  The listing
                               will not include the address and
                               machine code.
```

NOTES:

Output from this command will be directed the OUTPUT DEVICE if the L(ist) has been turned ON. Otherwise the listing will be displayed on the users screen.

If the command is terminated with a <:> or <;>, the dis-assembled output will appear as DATA statements only. This feature allows you to direct the dis-assembler to bypass normal mnemonic interpretation of machine code if you are certain that a block of memory contains only DATA or TEXT information. While the dis-assembler does not display TEXT statements, all DATA statements do include an ASCII translation of the data as a comment.

If the command is terminated with a <;> or <->, the output listing will not contain the address and machine code. It will produce an output listing that can be directly assembled if desired. This option is available for any output device, printer or disk file.

The Relocation address option allows you to copy a block of memory into a different area then disassemble it as if it were still in its original location. This feature then allows the addresses in the listing to reflect the original location of the block of code.

If you want to reassemble a source file that you dump to disk, you should first edit the source file to insert an AORG statement at the beginning of the file. The AORG should reflect either the area the program was disassembled from or the Relocation area where the program originated. This will make any branch instructions and data reference instructions correct when you try to run the re-assembled program. Also if the program uses any Editor/Assembler, Mini-Memory, or Extended Basic utilities, you should run the re-assembled program under the appropriate environment so the required utilities will be present.

If you want to disassemble a program for later reassembly, the following procedure is recommended: First dump the program to a printer using the D command. Then study the printout to find obvious TEXT and DATA areas of the program. Then use the A command to generate a disassembly to the printer using the <return> and <:> options. This will produce a first-pass disassembly with obvious TEXT and DATA areas disassembled into DATA statements.

Then review this first-pass listing to search for other DATA areas that may not have been obvious in the Dump. Some clues to such areas are a series of MPY, DIV, SZC, or SOC instructions. Many DATA values have a tendency to disassemble into these types of instructions. Another place to look for DATA values is immediately after a BL or BLWP instruction. Many times DATA values are passed to subroutines by such DATA statements. If one or more instructions following a BL or BLWP seem strange, this may be an indication that such DATA statements are being used. Another way to check for such DATA passing is to look at the code in the routine referenced by the BL or BLWP. If the BL routine uses any *R11+ operands or if the BLWP routine uses any *R14+ references, you can be sure that DATA is being passed to the routine in this manner.

Once you have identified all possible DATA and TEXT areas, use the A command once more to disassemble the program to disk using the <-> and <;> options. Hopefully you have jotted down a list of all the areas that need to be disassembled with the <;> option. You will have to disassemble each consecutive <-> and <;> area of the program with individual commands, but they can all be sent to the same disk file. The file will be written in APPEND mode so each successive A command will merely add source statements to the end of the existing file.

B -- BREAKPOINT SET/CLEAR

* To set a breakpoint

ENTER:
```
        B <Address> <return>
```

* To clear a breakpoint

ENTER:
```
        B <Address> <->
```

* To show breakpoints

ENTER:
```
        B <return>
```

* To clear all breakpoints

ENTER:
```
        B <->
```

Example:

```
        B 2480 <return>  Sets a breakpoint in the user's program
                         at location >2480.
```

NOTES:

Up to 11 breakpoints can be set at a time. SUPERBUG II uses two word breakpoints. When you set a breakpoint, the program inserts a branch to the breakpoint handler routine within SUPERBUG II. When the breakpoint is encountered during execution, the workspace pointer, program counter, and status are saved, the breakpoint is cleared by restoring the original code, and the SUPERBUG II breakpoint handler is entered. When the breakpoint returns to SUPERBUG II, the breakpoint encountered will be displayed and the program will be back in command mode waiting for your next entry.

Since SUPERBUG II uses 2-word breakpoints, you should observe the following precautions:

1) You cannot set breakpoints at consecutive words. If you attempt to do so, the following error message will appear:

```
              ILLEGAL CONSECUTIVE BRKPT
```

2) The same illegal consecutive breakpoint message will appear if you try to enter a breakpoint that already exists.

3) Be sure to set your breakpoints in a part of your program where the second word of the breakpoint is not referenced by some instruction in your program. If this happens, unpredictable results will occur.

4) When using breakpoints, it is always a good idea to type B- before exiting SUPERBUG II. This will help ensure that all breakpoints will be removed in case some were set but never reached during your testing. If you exit SUPERUG II with breakpoints still in your program, unpredictable results may occur if you try to execute the program.

Remember that you cannot set breakpoints in ROM-based programs. When a breakpoint is set, the original contents of the breakpoint location are copied into a temporary storage area within SUPERBUG II and a branch to the breakpoint handler is written into the program being tested. This cannot be done with ROM-based code so if you try to do this, the breakpoint will never be reached.

If you need to test ROM-based code, you should use the S command to single-step through the code rather than attempt to set breakpoints.

C -- CRU INSPECT/CHANGE

ENTER:

```
        C <Base Addr> <Bit Count> <return>
    then [<Data>] <return>
```

Example:

```
        C 1100,8 <return>  Gives the value of the first
                           8 bits at address >1100
```

NOTES:

Your first request causes SUPERBUG II to perform a STCR to read the number of bits specified. If you request 1 to 15 bits, that many will be returned. If you select 0 bits (or press return with no bit count), then 16 bits will be read and returned. After the initial display, SUPERBUG II is waiting for you to enter a new bit pattern (1 to 4 Hex digits) which it will then write to the specified CRU with a LDCR. Press return with no entry to return to the command mode.

The C command will cause SUPERBUG II to attempt to read from the specified bits. However, if the hardware does not support data input from the selected bits, the display will show a 1 or ON condition as if it had read an input. Similarly, if you attempt to write to a CRU bit where the hardware does not accept output from the computer, nothing will happen.

If you wish to examine the DSR ROM in a Peripheral Controller, use this command to enable the card and switch the ROM into the DSR address range from >4000 to >5FFF. Disk controllers are normally set to CRU address 1100, so the example above would be the first step for accessing a Disk Controller DSR. When a response is given, type the number 1 followed by a <return> to turn on Bit 0 of the CRU address.

Peripheral DSR ROMs are enabled by setting Bit 0 of the CRU address to a 1 and disabled by setting it to 0. Once the ROM is enabled, you can use the M command to examine its contents. If you want to Dump or Disassemble the DSR ROM to a printer or disk file, you must first use the N command to copy the ROM contents into a different area of CPU RAM. This is because only one device can be active in the DSR range at a time and when you try to direct output to disk or printer, its controller will deactivate the ROM you enabled and its contents will no longer be available. This is one of the reasons for the memory relocation features of the D and A commands.

Bit 0 is the only bit pre-defined in the TI specifications. Any other bits in the CRU address range for the peripheral are at the discretion of the designer. Refer to the technical documentation for the peripheral controller of interest for the meaning of any other CRU bits that it may use.

D -- DUMP MEMORY

* To Dump code at current User PC Location

ENTER:

```
        D {<return> or <-> or <:> or <;>}
    or D [<number>] {<return> or <-> or <:> or <;>}
```

Dumps the memory Data in HEX and ASCII beginning at the user PC Address. If a single paramenter is entered it is treated as the Number of locations to display or print.

Example:

```
        D 100 <return>   Dumps 256 (hex 100) bytes starting at the
                         user's current PC.
```

* To Dump code between locations specified

ENTER:

```
        D <Start Addr> <End Addr> [Relocation Addr>] <return>
```

Dumps the memory Data in HEX and ASCIIs beginning with the start address entered and continuing to the stop address entered.

Example:

```
        D A000,A080,6000 <return>   Dumps the memory from >A000
                                    to >A080 but the addresses
                                    displayed make the dump seem
                                    to be from >6000 to >6080.
```

NOTES:

Output from this command will be directed to the OUTPUT DEVICE if the L(ist) has been turned ON. Otherwise the listing will be displayed on the users screen.

The <-> termination for this command is similar to the <-> termination for the A command. If it is used here, the dump will not have the address and equal sign at the beginning of each line. This feature can be used for printer or disk file outputs.

The Relocation Address option for this command operates identical to the Relocation Address for the A command.

A G or V can be entered at the end of the Start Addr to indicate the dump should be made from GROM or VDP, respectively, rather than from CPU memory.

E -- EXECUTE

ENTER:
```
       E [<Address>] <return>
```

Example:
```
       E <return>  Begins execution at the PC address last
                   specifed by the R command.
```

NOTES:

If an optional Address is specified, this command will start execution at that address rather than the last address specified by the R command.

Before using this command, you should use the R command to at least initialize a workspace for your program even if you supply a PC address with this command. Failure to do so will cause unpredictable results because the workspace pointer is initialized to >0000 when the program is started which is console ROM. Thus registers cannot be changed unless you supply a workspace somewhere in RAM.

If you use breakpoints to halt execution of your program, or if you use S to single step through your program, SUPERBUG II will continually update the last PC entered via the R command as breakpoints are reached or as the S command is issued. If you enter E after a breakpoint or an S command, your program will begin execution from the last updated PC value.

The Q command can also be used to execute a user's program. Refer to the Q command description for details

F -- FIND WORD OR BYTE EQUAL

ENTER:
```
      F <Start Addr> <End Addr> <Data> [<return> or <->]
```

Example:
```
       F 2000,3000,FFFF <->  Searches from >2000 to >3000
                             comparing bytes to >FF and displays
                             the addresses and contents of those
                             bytes that match.
```

NOTES:

If the command is terminated with a <->, bytes are compared but if it is terminated with <return> then words are compared. When words are compared, the address is incremented by 2 after each comparison so the comparisons are always on word boundaries. If byte comparison is selected, the address is incremented by 1 after each comparison and each byte is compared to the specified data.

The K command is the opposite of the F command in that it will search for bytes or words that do not match the data specified. Refer to the description of the K command for more information.

If a G or V is entered after the Start Address, the search will be made in GROM or VDP memory rather than in CPU memory. Searches in these memory areas are always byte-oriented.

CTRL/F -- FIND STRING

ENTER:

```
      CTRL/F <Start Addr> <End Addr> <return>
```

Example:

```
     CTRL/F 2000,3000 <return>    Searches from >2000 to >3000
                                  comparing bytes to the current
                                  string defined by the I command
```

NOTES:

A special character combining an F and an S will be echoed when CTRL/F is pressed to indicate the Find String command.

The program will search all bytes in the specified range for matches with the current string. The string may be up to 10 characters in length. If the current string is 'ABCDEFGHIJ' and a match is found starting at location >2031, the following display will be presented:

```
      2031 = 4142434445464748494A
      2031 = A B C D E F G H I J
```

This two-line display shows both the hex values and the ASCII values for each byte in the string. This type of two-line display will be presented for each location in the range where a string match occurs.

The addresses may be byte or word addresses. The Start Address may be preceeded by a G or V to indicate that the search is to be performed in GROM or VDP memory.

G -- GROM BASE CHANGE

* To show the current GROM Base Address

ENTER:

```
      G <return>
```

Example:

```
      G <return>       Displays the following message:
                       GROM BASE = xxxx
                       to indicate the current GROM
                       base address in effect.
```

* To change the current GROM Base Address

ENTER a new address following the displayed GROM Base Address

Example:

```
      GROM BASE = 9800    9810 <return>   Changes the Page 0
                                          GROM Base Address of
                                          >9800 to >9810 for
                                          Page 4.
```

NOTES:

The TI-99/4A supports 16 pages of GROM addressing as follows:

GROM Page	Base Address
0	9800
1	9804
2	9808
3	980C
4	9810
5	9814
6	9818
7	981C
8	9820
9	9824
10	9828
11	982C
12	9830
13	9834
14	9838
15	983C

Be sure to only enter one of these specified Base Addresses or subsequent accesses to GROM from SUPERBUG II may not work properly.

The Base Address serves to select a particular bank of GROM/GRAM memory. Read/write control to GROM/GRAM is determined by adding an offset to the current base address. These offsets are >0402 for GRMWA (write address), >0400 for GRMWD (write data), and >0002 for GRMRD (read data). For example when SUPERBUG II reads from a GROM location (say G6000), it first writes the address >6000 to the GRMWA which is formed by adding >0402 to the current GROM base address. Then it reads from his location by adding >0002 to the current GROM base address to form the proper GROM Read Data Address and Moves Bytes from GROM until all desired bytes are read.

The GROM Base Address concept was intended to allow multiple command module programs to be housed in a single cartridge case. The 3 GROMs in the console which contain the operating system and TI Basic respond to all 16 pages so the Base Address has no effect on these GROMs. The paging only affects those GROMs in the cartridge memory space with addresses from G6000 to GFFFF. Each page of GROM memory supports up to 5 GROMs on 8K boundaries in this range. The GROM paging has no effect on CPU ROM or RAM in the cartridge space which resides in the CPU address range of >6000 to >7FFF.

If a cartridge does not contain multiple banks of GROM, the GROMs in the cartridge will respond to all Base Addresses.

The TI-99/4A console actually has a feature which supports startup menu display of multiple command modules. This feature is called REVIEW MODULE LIBRARY. If a multiple command module cartridge were installed, the startup screen would show the program selections for the Page 0 command module and an extra selection for REVIEW MODULE LIBRARY. If this extra selection is chosen, the startup screen would be redrawn showing the programs in Page 1 and the REVIEW MODULE LIBRARY selection. The startup screen would be redrawn showing the programs in the next page command module each time the REVIEW MODULE LIBRARY selection is made. When all available pages have been shown, Page 0 will be shown again.

TI never made any such multiple command module cartridges available as commercial products. However several manufacturers are currently working on products that will support GROM or GRAM paging using this technique. This feature of SUPERBUG II will make the program compatible with such hardware when it becomes available.

H -- HEXADECIMAL ARITHMETIC

ENTER:

```
      H <First Number> <Second Number> <return>
```

Example:

```
      H A 6 <return>  Displays the sum, difference, product,
                      quotient, and remainder of >A and >6.
```

NOTES:

Each operand must be a hexadecimal number with up to 4 digits. For the example given above, the resultant display will appear as follows:

```
H1=000A  H2=0006  H1+H2 = 0010
H1-H2 = 0004  H1*H2 = 0000 003C
H1/H2 = 0001  R = 0004
```

The first line shows the two numbers you entered along with the Hex Sum. The second line shows the Hex Difference and Product. Note that the product of two 16-bit words is a 32-bit word which is represented by two consecutive 16-bit words. The third line shows the Hex Division result. The Quotient and Remainder are each stored in a 16-bit word shown here as the Quotient followed by the remainder.

I -- ENTER STRING

ENTER

```
    I
```

NOTES:

This command allows you to enter the string that will be used for inserting strings into memory or searching memory for string matches.

After entering I, you will be presented with the following prompt:

```
     Insert/Search String:
     ABCDEFGHIJ
     ENTER STRING
     *
```

The string 'ABCDEFGHIJ' is the default string to indicate that up to 10 characters may be entered. Once a string has been entered, subsequent uses of the I command will present the last string that was entered.

To change the string, merely type the new string after the asterisk on the blank line and press ENTER when you are through. You may enter up to 10 characters. If you make a mistake, press FCTN/S for backspace and the prompt will be presented again.

You can actually enter any ASCII character that can be typed on the keyboard using the CTRL and FCTN keys as well as the alpha-numeric keys. Note however that some keys such as FCTN/S and FCTN/X will be treated as escape commands rather than string text. Many of the non-alphanumeric codes do not have a character pattern defined, so if you enter such codes, you will either see a blank or garbage character echoed on the screen. However, the hex code for the character is in the text string and the search will be performed properly.

CTRL/I -- INSERT STRING INTO MEMORY

ENTER:

```
     CTRL/I <Address> <return>
```

Example:

```
     CTRL/I 2800 <return>    Inserts the current string into
                             memory starting at >2800.
```

NOTES:

A special character combining an I and an S will be echoed when CTRL/I is pressed to indicate the Insert String command.

The last string entered with the I command (or the default string if the I command has not been used), will be inserted into memory beginning at the address specified by the CTRL/I command.

The address may be a byte for a word address and it may be preceeded by a G or V to indicate that the string is to be inserted into GRAM or VDP memory.

Remember that this feature only inserts the ASCII codes for the text string into memory. It does not insert a byte count for the string because this is not always necessary or desired. If you are replacing a string like a device pathname in a program that requires a string length byte in front of the string, you will need to use the M command to modify the length byte separately if your new string is not of the same length as the old one.

J -- TOGGLE SCREEN COLORS

ENTER:

```
        J
```

NOTES:

The screen color will be toggled to the next of 6 available foreground and background color combinations. Once the sixth choice has been used, pressing J again will cause the first color to be presented. The border color is set to match the background color.

The six choices available are:

WHITE on DARK BLUE	>F4F4
BLACK on LIGHT YELLOW	>1B1B
BLACK on LIGHT GREEN	>1313
BLACK on GRAY	>1E1E
BLACK on WHITE	>1F1F
DARK BLUE on GRAY	>4E4E

If you want to customize your copy of SUPERBUG II to change the order of thse colors or to replace them with different colors, search near the end of the SUPERBUG II program for the string of 6 words that matches the data values shown above for the default color combinations.

If you have a Disk Fixer program, you can use it to edit the sector containing these values so each time the program is loaded your customized values will be available. You will have to edit each of the 3 file versions of SUPERBUG II individually and the values will be in a different location in each.

If you only want to change them while the program is in memory, you can use the M command to find the values and change them. You should start searching about >80 bytes from the end of the program.

You can use SUPERBUG II itself to change the contents of the SBUG and SBUG6 files then use the Save Program File command to write the program back to disk with your modifications. Be sure to use the procedures described in the section on Patching Program Files and always work with a backup copy of the program.

If you make such modifications to your copy, they should be for your use only. If you give a copy of SUPERBUG II to someone else through FAIRWARE distribution, please be sure to give them a copy of the original distribution disk.

K -- FIND WORD OR BYTE NOT EQUAL

ENTER:

```
     K <Start Addr> <End Addr> <Data> [<return> or <->]
```

Example:

```
     K 2000,3000,FFFF <return>   Searches the memory from >2000
                                 to >3000 and displays those
                                 addresses and their contents
                                 that do not match >FFFF.
```

NOTES:

If the command is terminated with a <return>, words are compared but if the command is terminated with a <-> then bytes are compared.

Except for the fact that this command looks for data that does not match the user's entry, its operation is identical to that of the F command. Like the F command, the Start Addr can be followed by a G or a V to indicate that the search is to occur in GROM or VDP.

L -- LIST DEVICE TOGGLE

ENTER:

```
L
```

NOTES:

This command will toggle the List device between the Output device entered and the user screen display.

The current disposition is displayed on the screen. To change conditions, simply re-enter this command.

If you do not have a printer connected to your system, you should either define a disk file as your output device or use the L command to turn off the output device. This will cause D and A outputs to be issued to the screen rather than to a printer or disk file. If you accidentally try to send a D or A output to a printer that is not connected or turned on, the program may lock up and you may have to reset or restart your computer.

CTRL/L -- LOAD PROGRAM FILE

ENTER:

```
CTRL/L [<Load Addr>] <return>
```

NOTES:

A special character combining an L and a P will be echoed when CTRL/L is pressed to indicate the Load Program File command.

The Load Address entry is optional. If an address is specified, the program file will be loaded starting at that address. If no address is specified, the default address in the program file will be used.

After entering the CTRL/L command, the following prompt will appear:

```
Device/File:
DSK1.SAMPLE
ENTER FILE NAME
*
```

This prompt gives you an opportunity to change the default or last file pathname. To load the file indicated, merely press ENTER after receiving the prompt. To change the file name, type a new name of up to 28 characters following the asterisk then press ENTER. When you press ENTER, the current file name will be loaded. Following a successful load, the following message will appear:

```
Program starts at >xxxx
```

The actual start address of the program loaded will be displayed in place of the x's.

If you make a mistake in entering the file pathname, merely press FCTN/S for backspace and the prompt will be repeated. If you pressed CTRL/L by mistake and do not actually want to load a program file, press FCTN/X to exit the command routine.

If the program you wish to load is larger than 8K bytes and consists of several segments (such as DISK1.SAMPLE1, DSK1.SAMPLE2, etc.) you will have to load each segment individually because this command does not support automatic file name incrementing as is done when Editor/Assembler Option 5 is used. This was done deliberately to allow maximum flexibility with the CTRL/L feature.

M -- MEMORY INSPECT/CHANGE

* To see or alter a memory location

ENTER:

```
      M <Address> <return>
 then   [<Data>] {<return> or <->}
```

Example:

```
      M 3000 <return>   Causes the contents of location >3000
                        be displayed.  After the contents are
                        displayed, you may enter new data and
                        press <return> to change the data to
                        your new entry.
```

* To see multiple memory locations

ENTER:

```
      M <Start Addr> <End Addr> <return>
```

Example:

```
      M 2000,2080 <return>  Displays the address and contents
                            of locations >2000 to >2080.
```

NOTES:

If a range of addresses are specified, SUPERBUG II will display their addresses and contents on the screen. The screen can display up to 128 addresses or >80 at one time without scrolling any information off the screen. You can press any key to pause the function while it is scrolling then any key again to resume the scrolling. However if you press the FCTN and X keys simultaneously the command will be ended and SUPERBUG II will return to command mode.

If you enter a single address, its current contents will be displayed and SUPERBUG II will wait for your entry to change it or view the next address. If a change is followed by <return>, the change will be made and SUPERBUG II will return to command mode. If a change is followed by a <,> or a <space>, the change will be made and the next address will be displayed. If a <,> or <space> is entered without a change, the next address will be displayed without any change being made.

If you terminate a change with <->, the change will be made and the previous address will be displayed. This allows you to check the change you just made. If you enter <-> without making a change, the previous address will be displayed.

A G or V can be entered at the end of the Start Address to indicate that the data display should be from GROM or VDP respectively rather than from CPU RAM or ROM.

GROM cannot be altered, it can only be displayed. However, even accessing GROM changes its program counter so it may not be possible to return directly to your program from SUPERBUG II after examining GROM memory. If you have a GRAM emulation device, this command can be used to modify memory in the GRAM emulator.

Only addresses >0000 to >3FFF of VDP RAM can be accessed. These addresses may be examined or changed. The area in VDP RAM from >3500 or so to >3FFF is used for disk buffers by the Disk Controller. If you are doing disk output from SUPERBUG II, be careful not to alter any of these locations.

The 8 VDP write-only registers may be changed from SUPERBUG II but they cannot be examined. To change a VDP register, enter an address of V8 followed by the register number (0 to 7), the data byte, and <return>. For example:

```
        M V87F5
```

will load VDP register 7 with >F5.

N -- MOVE BLOCK

ENTER:

```
        N <From Addr> <To Addr> <Byte Count>
          {<return> or <-> or <:> or <;>}
```

Example:

```
        N 4000,C000,2000 <return>  Moves 8192 (hex 2000) bytes
                                   starting at >4000 to the
                                   block starting at >C000.
```

NOTES:

If the command is terminated with a <return> or <:>, the data will be transferred from the starting addresses specified to the ending address implied by the byte count. If the command is terminated with a <-> or <;>, the transfer will be in reverse order. In the example given above, if the command had been terminated by a <->, the first byte transferred would have been from >5FFF to >DFFF and the last byte transferred would have been from >4000 to >C000.

The byte transfer can be executed in fast or slow mode. To use slow mode, terminate the command with <:> or <;>. This will cause a delay of about 15 to 20 milliseconds between each byte transferred. This feature is provided mainly to allow a block of memory to be copied into EEPROM devices which require a minimum of 10 milliseconds between each write operation.

To program EEPROMs, you will need some type of hardware device that allows you to plug RAM or EEPROM chips into a socket somewhere in CPU memory space. The type of hardware you have available will determine the address range to use as the target for the EEPROM chips. Refer to the technical information for your hardware to determine the proper EEPROM addresses to use and whether any special setup logic is necessary before you can use the N command. For example, some hardware might require that you first use the C command to set certain CRU bits to enable the device before you can load data into it and program the EEPROM chips.

This command is also handy for initializing a block of memory. First use the M command to initialize the first word of the block then use the N command to initialize the rest of the block. To initialize >C000 to >CFFF to ZERO, first clear >C000 with the M command then use the N command as follows:

```
        N C000,C002,1000 <return>
```

O -- CHANGE OUTPUT DEVICE

ENTER:

```
        O
```

NOTES:

After entering O, you will be prompted for a new Output Device. The same prompt that appears when the program is first started will be displayed along with a blank line where you may enter a new output device name if you wish.

The previous output device name is displayed with the prompt. If you press <return> without making a new entry, the previous device name will be retained. If you make a new entry, it cannot exceed 28 characters in length.

As in the startup prompt, you can press FCTN and S simultaneously for a backspace to get the prompt back and start your entry again. After your entry is complete and you press <return>, a message will be displayed informing you that the List Device toggle is ON.

The Output Device name may be any file-oriented output device supported by the TI-99/4A system. In general this will either be a Printer or a Disk File. The Output Device will be used to print output from the D and A commands if the L option is set to enable the output device.

P -- COMPARE MEMORY BLOCKS

ENTER:

```
P <Start Addr 1> <Start Addr 2> <Byte Count> <return>
```

Example:

```
P 2000,3000,100 <return>   Compares the 256 (hex 100)
                           bytes from >2000 to >2100 to
                           those from >3000 to >3100 and
                           displays the addresses and
                           contents of those that do not
                           match.
```

NOTES:

A G or a V can be entered at the end of Start Address 1 or Start Address 2 to indicate that the comparison should be made with that block in GROM or VDP respectively rather than in CPU memory.

This command is useful for comparing different versions of a program. If you can load both versions of the program into memory at the same time, you can compare the contents of both memory areas where the programs reside to see how extensive the changes were from one version to the next.

Another valuable use of this feature is memory tests. If you have SUPERBUG II loaded into SUPER SPACE for example, you can initialize all of CPU memory from >2000 to >3FFF and >A000 to >FFFF to some value (say >0000, >FFFF, >AAAA, or >5555). These values are suggested because they check all zeroes, all ones, and alternating ones and zeroes.

You can then use the P command to compare the contents of the low memory expansion block from >2000 to >3FFF to each of the three 8K blocks in high memory one at a time. If you compare all blocks with the 4 patterns indicated, you will perform a fairly extensive check on the read, write, and refresh capability of your expansion memory.

This same feature can be used to check 128K to 512K expansion memories also. You will probably have to use the C command in between testing each 32K block though to switch the next bank into memory. Refer to the technical literature provided with your hardware to determine the proper way to run such tests.

Q -- QUIT SUPERBUG II

* To Exit SUPERBUG II and execute a user program

ENTER:

```
Q [<Address>] <return>
```

* To Exit SUPERBUG II and restart the computer

ENTER:

```
FCTN/= simultaneously
```

NOTES:

The Q option is similar to the E command except that Screen Address, Offset, and Width are reset prior to Executing the User's program. The PC to which the Q option branches is determined in the same manner as for the E command.

The FCTN/= option causes a BLWP to location >0000 which causes a computer restart. If you have a TI-99/4 console, use SHIFT/Q instead of FCTN/=.

R -- INSPECT OR CHANGE WP, PC, AND SR

ENTER:

```
       R <return>
  then  [<Data>] [<return> or <space>]
  then  [<Data>] [<return> or <space>]
  then  [<Data>] [<return> or <space>]
```

NOTES:

This command allows you to inspect and change the WP (Workspace Pointer), PC (Program Counter), and SR (Status Register) that will be used for your program when you select the E, Q, S, or V commands.

When you first select R, SUPERBUG II will display the value of the WP. If you change it then press <return>, the value of the PC will be displayed. If you press <return> without making a change, the command will terminate and you will be returned to the SUPERBUG II command mode. If you wish to advance to the PC without making a change, merely press <space>.

Once you have viewed or changed the PC, the same procedure will advance you to the SR value. After viewing or changing the SR, you will be returned to the command mode of SUPERBUG II.

The WP points to the workspace for your program, the PC points to the first instruction that will be executed, and the SR sets up the initial values for the Status Register. SUPERBUG II uses these values as "return" parameters to enter your program with a RTWP instruction when you select E or Q. If you select S or V, SUPERBUG II will execute your program in "interpretive" mode using these values as the environment for your program.

S -- SINGLE STEP

ENTER:

```
       S
```

NOTES:

The instruction located at the user's PC is executed and displayed on the screen next to it's memory address. The effective jump address is also shown for all JUMP instructions.

Care must be taken when stepping through VDP RAM accessing. You should avoid this by placing a breakpoint following the VDP access instructions. You should also avoid stepping through GROM code.

Since the S command executes your program interpretively, it can be used to single step through ROM or RAM code. The next instruction pointed to by the PC value is fetched from your program, copied into SUPERBUG II RAM space, and executed there. If the instruction is a 2-word instruction, both words are copied into SUPERBUG II RAM space before executing the instruction.

Caution should be exercised if you use this feature to execute console or DSR ROMs. Unpredictable results may occur, the computer may lock up, and if you are stepping through a Disk DSR you may mess up a diskette.

CTRL/S -- SAVE PROGRAM FILE

ENTER:

```
      CTRL/S <Start Addr> <End Addr> [<Load Addr>] <return>
```

Example:

```
      CTRL/S C000,E000 <return>    Saves the memory image from
                                   >C000 to >E000 in a PROGRAM
                                   file format.
```

NOTES:

A special character combining an S and a P will be echoed when CTRL/S is pressed to indicate the Save Program File command.

The Load Address is optional. If one is specified, the default address in the PROGRAM file will be set to the Load Address otherwise the Start Address will be used as the Load Address.

After entering the CTRL/S command, the following prompt will appear:

```
     Device/File:
     DSK1.SAMPLE
     ENTER FILE NAME
     *
```

This prompt gives you an opportunity to change the default or last file pathname. To save a program to the file indicated, merely press ENTER. To change the file name, type a new name of up to 28 characters following the asterisk then press ENTER. When you press ENTER, the memory area will be saved.

This command supports automatic file pathname incrementing. Therefore if you entered a file name of DSK1.SAMPLE1 but the memory image is 10K bytes in length, part of the memory image will be saved to DSK1.SAMPLE1 and part will be saved to DSK1.SAMPLE2.

Memory images saved in this manner are compatible with Option 5 of the Editor/Asembler. It is important to ensure that the Start Address of your memory image actually contains the first executable instruction of the program area if you plan to load the program file with Option 5 of the Editor/Assembler.

T -- TRADE SCREEN (BIT MAP MODE ONLY)

* To toggle or Trade the user screen for the SUPERBUG II screen

ENTER:

```
      T
```

NOTES:

The SUPERBUG II screen will be swapped with the user screen ONLY if BIT MAP mode was selected upon entry to SUPERBUG II. Each time the <T> is pressed, the screen will toggle from user screen to SUPERBUG II and reverse.

SUPERBUG II maintains two VDP register data arrays, one for its own use and one to match the loading environment. When BIT MAP mode is selected, these two arrays are set differently to support the T command. When BIT MAP is not selected, both arrays are set the same (to match E/A, X-B, or Console Basic environments). If the T command is used when BIT MAP mode is not selected, the VDP registers are still toggled between the two arrays but since they are identical, there is no perceptive change.

U -- TOGGLE OFFSET TO AND FROM TI BASIC

ENTER:
    U

NOTES:

This command changes the screen offset by plus or minus >60 to allow you to alternate displaying the screen as TI BASIC uses it and as SUPERBUG II (or most Assembly Language programs) use it. TI BASIC (and Extended BASIC) offset the ASCII characters in VDP RAM by >60 from the way in which they are normally stored by most Assembly Language programs.

The reason for this offset is to compact the VDP storage area as much as possible to reserve the maximum possible amount of VDP memory for Basic programs. The screen area uses VDP memory from >0000 to >02FF. Character tables occupy 2K bytes on >800 byte boundaries. Most assembly language programs use the first such boundary after the screen which is from >800 to >FFF.

Basic sets the character pattern table from >000 to >7FF to conserve space. The first >300 bytes of the character patterns overlap with the screen memory, so the first 96 characters (0 to >5F) are not available. The first character pattern available is at >300 which is character number >60. To treat this character as character number ">00" requires adding an offset of >60 to obtain the proper character pattern address.

V -- EXECUTE UNTIL VALUE EQUAL

ENTER:
    V <Address> <Value> <return>

NOTES:

This command is similar to the S command in that it copies instructions from the user program into SUPERBUG II RAM for execution. The main difference is that instead of returning to SUPERBUG II command mode after each instruction, it continues to execute the user's program.

The user program is executed in slow mode until the value contained at the address specified is equal to the value entered. When this occurs, SUPERBUG II will return to the command mode with the message "BP NOT FOUND" to indicate that the halt was due to a value match rather than a breakpoint.

It is recommended that you do not try to execute console operating system code using this command. If your program makes any branches into System utilities such as SCAN or GPLLNK routines, then results are unpredictable and your computer may lock up requiring you to power OFF then back on to recover.

This option should only be used for executing code which will not leave your program area.

W -- INSPECT OR CHANGE REGISTER

ENTER:

```
        W [<Register Number>] <return>
  then    [<Data>] [<return> or <space> or <->]
          (...)
```

Example:

```
        W <return>  Displays all of your Workspace Registers
                    and their values.
```

NOTES:

If you merely enter W and <return>, the value of all your Workspace Registers will be displayed and SUPERBUG II will return to command mode. The Workspace Registers displayed are those at the workspace specified by the WP in the R command.

If you enter a Register Number, the value of that register will be displayed and you may change it if desired. In this mode, the operation of the W command is similar to the M command. Pressing <return> will return to command mode, pressing <space> will advance to the next register, and pressing <-> will display the previous register again.

X, Y, Z -- CHANGE BIAS

* To Change X bias

ENTER:

```
        X <Value> <return>
```

* To Change Y bias

ENTER:

```
        Y <Value> <return>
```

* To Change Z bias

ENTER:

```
        Z <Value> <return>
```

Example:

```
        Z 6A <return>  Changes the Z bias to >6A.
```

NOTES:

This feature allows you to change the X, Y, or Z bias to the value entered. After a value is entered, you use the characters X, Y, and Z following an address in any other command and alter that address by the amount of the bias. If the result of the alteration is an odd value, the actual address displayed is the next lower value.

In the example above, if the M command were used as follows, SUPERBUG II would display the value of location >106A:

```
        M 1000Z <return>
```

> -- HEXADECIMAL TO DECIMAL CONVERSION

ENTER:

```
    > <Hex Value> <return>
```

Example:

```
        > 2C <return>  Displays the decimal value of >2C which
                       is 44.
```

NOTES:

This command displays the decimal value of a hexadecimal number with up to 4 digits. Values from >8000 to >FFFF are treated as negative numbers and interpreted as two's-complement binary values for the conversion to decimal.

Only integer decimal values are displayed. This routine does not support conversion to floating point (real) decimal values.

. -- DECIMAL TO HEXADECIMAL CONVERSION

ENTER:

```
        . <Decimal Value> <return>
```

Example:

```
          . 31 <return>  Displays the Hexadecimal
                         equivalent of 31 which is >1F.
```

NOTES:

This command displays the hexadecimal value of any decimal integer from -32768 through 65535.

The computer can represent a total of 65,536 unique numbers in 16 bits which is represented in Hex from 0000 to FFFF. Values from 0 to 32767 are converted to Hex numbers from 0000 to 7FFF.

Negative numbers from -1 to -32768 are converted to two's complement form and represented as FFFF to 8000. This range of Hex numbers maps into the same space as is used for positive numbers from 32768 to 65535 which is represented by Hex numbers 8000 to FFFF.

FINDING YOUR PROGRAM IN MEMORY

If you loaded your program from a tagged object file, the entry point of your program will be in the REF/DEF table if you used a DEF statement to define the entry point when you assembled the program.

The REF/DEF table starts at >3FFF and works its way downward. Each entry in the table occupies 8 bytes. The first 6 bytes are the program name and the last word (2 bytes) are the entry point of the program or routine.

Use the M or D option to examine or dump the area containing the REF/DEF table to find the entry point of your program. The address for your program's entry point will give some clue as to where it is loaded.

When you use the Load and Run option of Editor/Assembler, it begins loading programs at >A000 if they are relocatable (no AORG statement). The SBUGO file is relocatable so it may be loaded either before or after your program.

If you used the Extended Basic loader to load your program, it begins loading at >24FA. If you loaded your program before LOADSBUG, your program will start at >24FA if it is relocatable. If you loaded LOADSBUG before your program, your program should be 618 bytes past >24FA. If you use an AORG to put your program into High Memory, be sure to not use the area from >A000 to >BFFF because this is where LOADSBUG will load SUPERBUG II.

Once you enter SUPERBUG II and are ready to start testing your program, be sure to use the <R> command to set the WP, PC, and SR values before you try to use the <S>, <E>, <Q>, or <V> commands. Prior to using the <E> command, be sure to use the <B> command to set a breakpoint in your program at a place you expect it to reach so it will stop and return to SUPERBUG II.

If you forget to set breakpoints before using the <E> command, you will probably not be able to return to SUPERBUG II without resetting your computer and reloading your program and SUPERBUG II. Breakpoints are not necessary when using the <S> and <V> commands.

LOADING/SAVING PROGRAM FILES

Most object files loaded with Option 3 of the Editor/Assembler do not automatically start executing. Once they are loaded, control returns to the load prompt so you can load additional files or start running a program by pressing ENTER to advance to the Program Name prompt. However, all program files loaded with Option 5 of Editor/Assembler start running automatically. This makes it difficult to debug or disassemble such programs.

The Load/Save Program File features overcome this problem because now the entire operation can be controlled by SUPERBUG II. The Load command can be used to load 8K byte segments of the program file into either their normal memory areas or into any memory area you wish. If you want to execute or single-step the program, it should be loaded into its normal memory area or address references will be incorrect and the program will not run properly. However, if you merely want to dump, disassemble, or patch the program, it can be loaded into any convenient area of CPU memory. In such cases, the various address relocation features of SUPERBUG II can be used quite handily.

The Load feature does not automatically increment the file name and load subsequent segments if the program file is greater than 8K bytes. In such cases you will have to repeat the CTRL/L command if you want to load all segments. This was done so that large program files could be dumped, disassembled, or patched segment by segment.

The Save Program File feature is more convenient to use than the SAVE routine supplied with the Editor/Assembler. The E/A SAVE utility requires that you first load an object file that was assembled with the labels SFIRST, SLAST, and SLOAD DEFined. If you have an object file without these labels, you can now save it with SUPERBUG II.

The procedure for saving such files is to first load the object file you want to save with Option 3 of Editor/Assembler. Then load and run SUPERBUG II, determine the address range of the program you want to save, and use the CTRL/S feature to save the program.

Another advantage of the SUPERBUG II Save routine over the E/A SAVE utility, is that you can now save programs that occupy areas of low memory where the E/A SAVE program has to reside. The E/A SAVE is only intended to save programs loaded into High Memory from >A000 to >FFFF.

The SUPER SPACE version of SUPERBUG II is the best one to use for Loading and Saving program files. This version makes the task easier in many cases because when SUPERBUG II is in the cartridge memory space from >6000 to >7FFF, it probably will not interfere with address ranges for the program you want to load or save. This will avoid the need to use the relocation features.

PATCHING PROGRAM FILES

When a program file is loaded into memory with the CTRL/L command, it can be patched and written back to disk with the CTRL/S command. This eliminates the need to use a Disk Fixer program and trying to figure out sector values and offsets.

The easist things to change in a program are text strings such as default file or printer names. These are usually the first things that most people want to change if a program does not provide convenient ways to change them.

To change this type of data, use the I and CTRL/F commands to find the default string in memory. Once you have found it, use the I and CTRL/I commands to replace the program's default string with your own. If you are changing a printer or file name, you may have to adjust the byte length for the string which is usually (but not always) located just before the first character for the string.

When making such patches, be sure to examine the area of the program where the default name is stored. If your name is shorter than the default, you should not have any problems but if your name is longer, be sure there are enough blank spaces after the default name for your name. If there is not, do not overwrite other data or instructions that the program needs to use. In such cases, search the program for all references to the string name or length byte address and change them to an unused area of memory and insert your string into this unused area.

After making all your changes, use the CTRL/S command to Save the program back to disk. Always work with a backup copy of your distribution program in case you make a serious mistake that no longer lets the program run properly.

Another good practice is to use the D or A commands to dump or disassemble the program you want to change. Keep these printouts for reference in case you need to restore your patches. Also keep a record of your changes so you can restore the program to its previous state if necessary.

If you want to make such changes to SUPERBUG II, you should not try to patch and Save from within the same program. For example, do not load the SUPER SPACE version, make in-place patches then try to use the Save feature to copy the memory image back to SBUG6. Instead, use the SBUG or SBUGO versions to Load SBUG6, make patches to it, and Save it back. Similarly, use SBUG6 or SBUGO to Load, patch, and Save the SBUG version. If you want to patch the SBUGO file you should use a Disk Fixer program because this file is in Tagged Object format and is not compatible with the Load and Save features of SUPERBUG II.