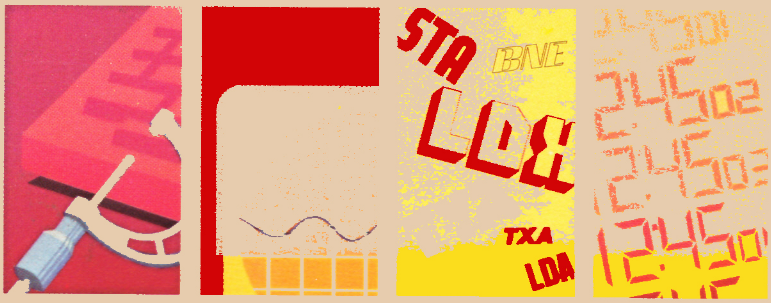
Bug Byter



The Screen-Oriented 6502-Debugger

Bug Byter

programmed by TED COHN
design and documentation by PETE ROWE

Disclaimer of All Warranties and Liabilities

ACKNOWLEDGMENTS

A bevy of special folks helped us in the design and implementation of BUGBYTER and this manual. Their unselfish support deserves recognition.

Special thanks to Tim Aaronson for both design recommendations and technical assistance.

Much gratitude for Geoff Zawolkow's assistance with design and documentation and Lawrence You's helpful suggestions.

Finally, our thanks to Carol Rowe and Arlan and Joan Cohn for their general support and patience.

Even the earliest computers had their problems, and even then they were called "bugs." According to Navy Captain Grace Murray Hopper, a pioneer in computer technology during World War II, the first computer "bug" was discovered at Harvard in August 1945.

Hopper, 74, said she and her associates were working on the Mark I, which she affectionately calls "the granddaddy of today's computers."

"Things were going badly; there was something wrong in one of the circuits of the long, glass-enclosed computer," she said recently. "Finally, someone located the trouble spot and, using ordinary tweezers, removed the problem -- a two-inch moth. From then on, when anything went wrong with a computer, we said it had bugs in it."

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Computers of larger size and complexity than Apple personal computers have static front-panel displays to report to the operator on the current status of the Central Processing Unit (CPU). They usually contain a few banks of binary lights that display just a few registers. Since the Apple and other personal computers are not blessed with such an array of lights, we conceived and designed Bugbyter as a customizable front-panel display for the Apple II and Apple /// computers.*

Bugbyter is a relocatable, 6502 machine language, mnemonic debugger. It features a user-definable display, literal and transparent breakpoints, a resident assembler and disassembler, and is compatible with Apple DOS.

A major design criterion was to model Bugbyter's commands after the Apple's Monitor (F8 ROM) command set. This forced-compatiblity should enhance comprehension and acquisition of Bugbyter's command language by a user already familiar with the Monitor ROM.

Another major design goal was to create a screen-oriented, user-definable display optimized for the 40 by 24 character Apple screen. All of the Apple's 6502 internal registers, a user-definable portion of the stack, mnemonic disassembly, user-selected memory cells and breakpoints and the command line are displayed on one Apple text screen. This has three notable effects:

(1) At any given time while debugging a program, a Bugbyter user has complete information on the status of the 6502 registers and stack.

(2) Since Bugbyter displays all information via absolute screen addressing, all input and output through Apple's I/O hooks (CSW and KSW) are unimpeded.

(3) Due to the screen-oriented display and non-I/O hook screen addressing, Bugbyter is not suited for serial, scrolling output to a printer.

A third major goal in the design of Bugbyter was to minimize memory contention. Bugbyter resides in approximately 6000 ($1A00) bytes of contiguous RAM, anywhere in the Apple's RAM from just above the text screen ($800) to just below the Monitor ($F800). Bugbyter also needs some zero page memory. However, the contents of these zero page cells are saved prior to Bugbyter's use and then restored, thereby eliminating any zero page contention. Bugbyter does use the first 32 bytes of the 6502's stack ($100 to 11F) and is unable to save or restore them. Any attempt of the user's program to alter the beginning of the stack could result in a collision between Bugbyter and the program being traced. Bugbyter flashes a warning at the displayed stack pointer when a user or user's program moves the stack pointer anywhere between $100 and $11F as discussed in the next chapter and Appendix A.

* Apple is a registered trademark of Apple Computer, Inc.

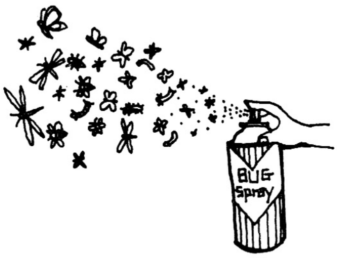
The last major feature added to Bugbyter prior to the production of this manual was real-time subroutine execution. Any 6502 code contained in a user-specified region that gets called as a subroutine will execute at the full speed of the processor. Therefore, any code that,

(1) need not be traced -- for example, the Apple Monitor character display code, or

(2) needs to execute in real-time -- for example, the low-level disk routines and paddle A to D conversion code

can execute at the full speed of the Apple's 6502 CPU.

Bugbyter is provided on a standard, 13-sector format, unprotected 5 1/4" diskette that will boot on a 48K Apple II or Apple II plus, 13- or 16-sector format. The Bugbyter diskette will also boot on an Apple /// using the standard Emulation diskette. All the files on the Bugbyter diskette can be moved to another 13-sector or 16-sector diskette (see next chapter and/or an Apple DOS 3.2 or 3.3 Reference Manual).



CHAPTER II

HOW TO USE THIS MANUAL

It is possible to use Bugbyter without reading this entire manual. The next chapter, "A Quick Tutorial," contains enough information necessary for minimal operation of Bugbyter. Later, you can assimilate more Bugbyter commands and functions by referring to specific chapters in this manual. We highly recommended that any learning of Bugbyter occur at the keyboard of your Apple.

NOMENCLATURE USED IN THIS MANUAL

All addresses are hexadecimal unless otherwise stated. Sometimes the manual includes a hex modifier -- sometimes not. Therefore, $5FF is equivalent to 5FF -- which is especially true for the Bugbyter Master display.

In general, words that are in all capitals, for example, RETURN, CTRL, B, ESC, refer to specific keys on the Apple's keyboard or to a Bugbyter command like SET or MEM.

This manual differentiates between the Apple's "screen" (lores, hires and text) and Bugbyter's "displays" (Master, SET and memory page).

BACKUP

The Bugbyter diskette is not protected. This will allow you to backup Bugbyter program files on another diskette. Use a 13-sector FID (FIle Developer) to move Bugbyter to another 13-sector, preformatted diskette, or MUFFIN to move Bugbyter files to a preformatted, 16-sector diskette. Both FID and MUFFIN are Apple utility programs normally distributed on the Apple DOS 3.3 System Master and discussed in Appendices J and K of the accompanying DOS 3.3 Manual. FID is normally distributed as a 16-sector program, but is quite capable of operating in a 13-sector environment.

The original Bugbyter diskette is a 13-sector diskette "updated" by the addition of a 16-sector boot sector added to allow for booting in both 13- and 16-sector environments. This makes whole diskette backups impossible. Therefore, please use the above-mentioned backup procedures.

Bugbyter is a tool for experienced programmers and a learning aid for aspiring programmers. We recommend any of the following documents for reference and introduction to the 6502 microprocessor for either kind of user.

REFERENCE

Programming Manual, MCS6500 Microcomputer Family, 1976, MOS Technology, 950 Rittenhouse Rd., Norristown, PA 19401. Pub. #6500-50A.

[The standard reference for programming the 6502 by the company that designed that microprocessor].

6502 Microprocessor Instant Reference Card, 1980, Micro Logic Corp., POB 174, Hackensack, NJ 07602. Product #101A.

[An excellent, single-card chart of everything you want to know about programming the 6502].

Applications Information SY6500 Microprocessor Family, 1980, Synertek Inc., POB 552-MS/34, Santa Clara, CA 95052.

[An in-depth pamphlet on 6502 internal operation including complete opcode timing diagrams].

GUIDES

Lance Leventhal, 6502 Assembly Language Programming, 1979, Osborne McGraw-Hill, 630 Bancroft Way, Berkeley, CA 94710.

[A quite complete guide to programming the 6502, 6520 and 6522].

Roger Wagner, "Assembly Lines," Softalk Magazine. Softalk Publishing, 11021 Magnolia Blvd., N. Hollywood, CA 91601.

[An on-going column devoted to programming the 6502 in the Apple computer].

CHAPTER III

A QUICK TUTORIAL

This chapter will introduce a subset of Bugbyter commands. What follows is a concise tutorial to get you up and running quickly. For more complete operating instructions, refer to Chapters IV through XI.

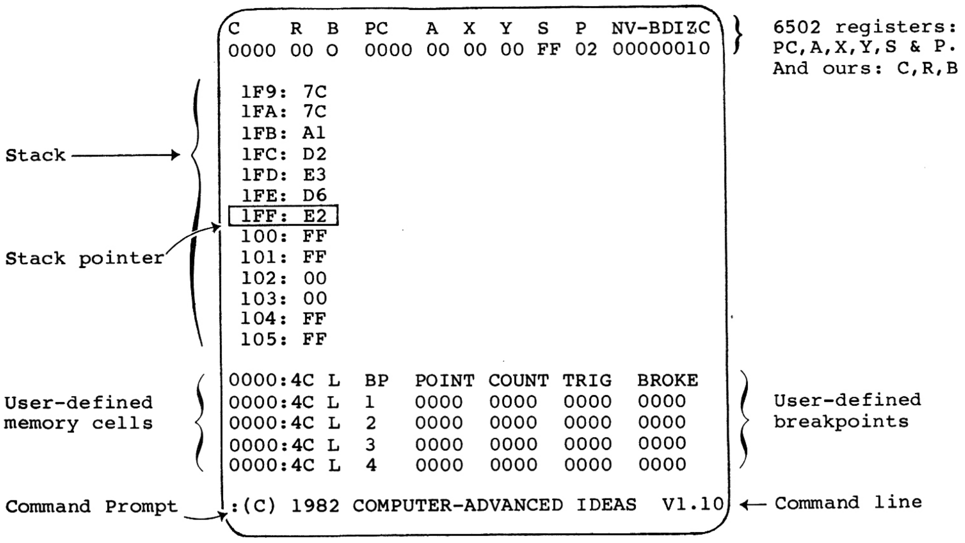
(1) Boot the Bugbyter diskette.

(2) At any time, press the ESC key to stop the animated title frame.

(3) Type:

```
BRUN BUGBYTER and press RETURN
```

(4) Your Apple Screen will now show the Bugbyter Master Display:



```
C    R  B  PC   A  X  Y  S  P  NV-BDIZC
0000 00 0  0000 00 00 00 FF 02 00000010

 1F9: 7C
 1FA: 7C
 1FB: A1
 1FC: D2
 1FD: E3
 1FE: D6
 1FF: E2
 100: FF
 101: FF
 102: 00
 103: 00
 104: FF
 105: FF

0000:4C L  BP  POINT COUNT TRIG BROKE
0000:4C L  1   0000  0000  0000 0000
0000:4C L  2   0000  0000  0000 0000
0000:4C L  3   0000  0000  0000 0000
0000:4C L  4   0000  0000  0000 0000

:(C) 1982 COMPUTER-ADVANCED IDEAS  V1.10
```

6502 registers: PC,A,X,Y,S & P. And ours: C,R,B
Stack
Stack pointer
User-defined memory cells
User-defined breakpoints
Command Prompt
Command line

(5) Type:

```
.BLOAD HIRES EXAMPLE and press RETURN
```

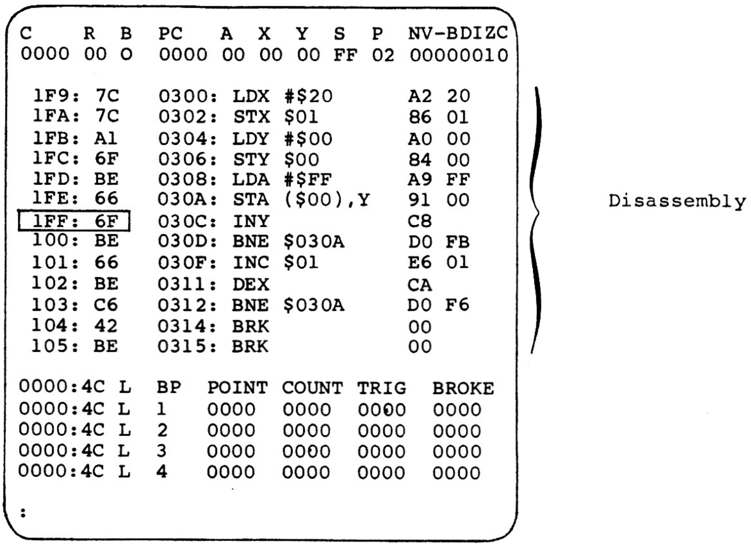
(6) Press RETURN again

(7) Type:

```
300L and press RETURN
```

You have now loaded two sample routines (from one file: HIRES EXAMPLE).

(8) Your Bugbyter Master Display should now look similar to:



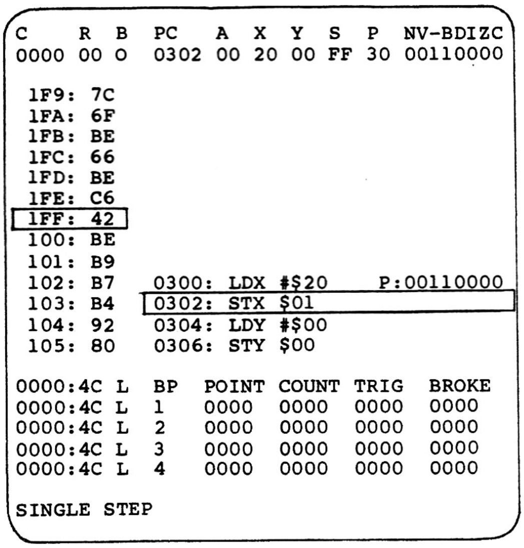
```
C    R  B   PC   A  X  Y  S  P   NV-BDIZC
0000 00 0   0000 00 00 00 FF 02  00000010

 1F9: 7C    0300: LDX #$20       A2 20
 1FA: 7C    0302: STX $01        86 01
 1FB: A1    0304: LDY #$00       A0 00
 1FC: 6F    0306: STY $00        84 00
 1FD: BE    0308: LDA #$FF       A9 FF
 1FE: 66    030A: STA ($00),Y    91 00
 1FF: 6F    030C: INY            C8
 100: BE    030D: BNE $030A      D0 FB
 101: 66    030F: INC $01        E6 01
 102: BE    0311: DEX            CA
 103: C6    0312: BNE $030A      D0 F6
 104: 42    0314: BRK            00
 105: BE    0315: BRK            00

0000:4C L  BP  POINT COUNT TRIG  BROKE
0000:4C L  1   0000  0000  0000  0000
0000:4C L  2   0000  0000  0000  0000
0000:4C L  3   0000  0000  0000  0000
0000:4C L  4   0000  0000  0000  0000

:
```

9) Type:

300S and press RETURN

You are now in Single-Step mode. Bugbyter will replace the disassembly subdisplay with the first three instructions of the sample routine, with the first instruction highlighted (inverse characters) and "SINGLE STEP" displayed at the bottom of the screen. The LDX #$20 is yet to be executed.

(10) Press SPACE. The PC (the program counter displayed at the top of the screen) is now set to 302; the X and P registers now reflect the LDX #$20 instruction just executed. Refer to the screen image on the next page.



```
C    R  B  PC   A  X  Y  S  P  NV-BDIZC
0000 00 0  0302 00 20 00 FF 30 00110000

 1F9: 7C
 1FA: 6F
 1FB: BE
 1FC: 66
 1FD: BE
 1FE: C6
 1FF: 42
 100: BE
 101: B9
 102: B7   0300: LDX #$20     P:00110000
 103: B4   0302: STX $01
 104: 92   0304: LDY #$00
 105: 80   0306: STY $00

0000:4C L  BP  POINT COUNT TRIG  BROKE
0000:4C L  1   0000  0000  0000  0000
0000:4C L  2   0000  0000  0000  0000
0000:4C L  3   0000  0000  0000  0000
0000:4C L  4   0000  0000  0000  0000

SINGLE STEP
```

(11) Press: RETURN

You are now in Trace mode. Bugbyter will replace the words SINGLE STEP with TRACE at the bottom of your screen and begin tracing.

(12) Press: H

You will now be viewing Apple hires screen page one and observing our example routine slowly white washing the hires graphics screen one row at a time.

(13) Press: T

You will return to the Bugbyter Master Display (Apple text screen).

At any time in Trace/Single-Step mode, you can press the SPACE bar to enter Single-Step mode. Pressing the RETURN key, at any time, will return you to Trace mode.

(14) Press: ESC

Now you are out of Trace/Single-Step mode and back into Bugbyter command mode.

(15) Press: Q RETURN

You will return to Basic+DOS

TUTORIAL SUMMARY

	Mode (Environment)	
Basic+DOS	:Bugbyter Command	Trace/Single-Step
(boot Bugbyter diskette)		
]BRUN BUGBYTER RETURN		
	.BLOAD HIRES EXAMPLE RETURN	
	300L RETURN	
	300S RETURN	
		SPACE
		RETURN
		H
		T
		ESC
	Q RETURN	
]		

In HIRES EXAMPLE still in your Apple's RAM, at $316 exists another small, sample routine that turns hires page one to black. Experiment tracing this code or just use it to clear the hires page by typing:

316G and press RETURN

For a complete description of Bugbyter operations and functions, read Chapters IV through XI. For a complete summary of Bugbyter commands, refer to Appendix A.

CHAPTER IV

OPERATION

Bugbyter is 6.7K ($1A00), binary DOS file. To execute Bugbyter on a 48K+ Apple, from either Basic type:

```
BRUN BUGBYTER  RETURN
```

At this time your disk drive will turn on, Bugbyter will load from $7C00 to $95FF and your screen will contain the Bugbyter Master Display (see next chapter for description of Master Display). The default starting address, $7C00, was chosen to locate Bugbyter in the highest memory just below DOS buffer one. Bugbyter, however, is relocatable and therefore can be loaded and run at any address in memory with only minor restrictions.*

To select your own starting address, for example, $21B3, from either Basic, type:

```
BRUN BUGBYTER,A$21B3   RETURN
```

Bugbyter will load from $21B3 to $3BB2. To use Bugbyter in a language or RAM card, type the following:

```
BRUN BUGLOADER  RETURN
```

which will automatically load and execute Bugbyter in your language card starting at $D000. Running this little program, BUGLOADER, is equivalent to manually typing from either Basic:

```
CALL -151  RETURN

C081 C081 F800<F800.FFFFM C083 C083  RETURN

BRUN BUGBYTER,A$D000  RETRUN
```

Bugbyter has its own variable storage, so the language card must not be write-protected (the C083 C083 above). $D000 is the Bugbyter starting address fixed in BUGLOADER. But Bugbyter can actually start at any address in a language card with only the previously mentioned limitations (see footnote this page*).

STARTING SUMMARY. Bugbyter will default start at $7C00. A user may specify any starting address from $800 to $7C00 (or up to $A600 if DOS is not required). Bugbyter may also be used in a language card from $D000 to $DE00. (Refer to Appendix B for an Apple memory map and a condensed list of specifications).

* Memory restrictions: Bugbyter cannot begin in memory less than $800, since that would result in placing Bugbyter program code into text screen memory ($400-$7F7). Likewise, the $C000-CFFF space is reserved for Apple I/O and peripheral memory. And memory above $F800 must contain the old or new auto-boot Apple Monitor.

Restarting. Once Bugbyter is loaded (BLOADed or BRUN) into memory, you can use the load address to start or restart execution of Bugbyter.

Example:

```
BLOAD BUGBYTER,A$2000   RETURN
```

You can now start or later restart Bugbyter from either Basic with:

```
CALL 8192   RETURN

      or

CALL 1016   RETURN       (page 3, CTRL Y vector)
```

Or from the Monitor with either:

```
2000G  RETURN

      or

CTRL  Y  RETURN
```

BLOADing vs. BRUNning. As mentioned in the Introduction, Bugbyter is a non-protected program. You are encouraged to backup your Bugbyter files. The approved method of transferring a Bugbyter file is:

```
BLOAD BUGBYTER,A$7C00   RETURN
```

Insert a format equivalent diskette (13- or 16-sector) and type:

```
BSAVE BUGBYTER,A$7C00,L$1A00   RETURN
```

Note that you did not BRUN Bugbyter first. BRUNning Bugbyter results in execution of a self-modifying sequence that fixes it to the address it was BRUN at. For example, if you typed:

```
BRUN BUGBYTER,A$1234   RETURN

:Q  RETURN                         (from inside Bugbyter)

BSAVE NEWBUGBYTER,A$1234,L$1A00   RETURN
```

You would create a Bugbyter program file (NEWBUGBYTER) that could only be BRUN at $1234 from now on. In other words, the self-modifying sequence that allows Bugbyter to be relocatable is a one-shot, non-reversible operation.

CHAPTER V

MASTER DISPLAY

The Bugbyter Master Display is divided into six subdisplays:

(1) Registers: 6502 and Bugbyter.

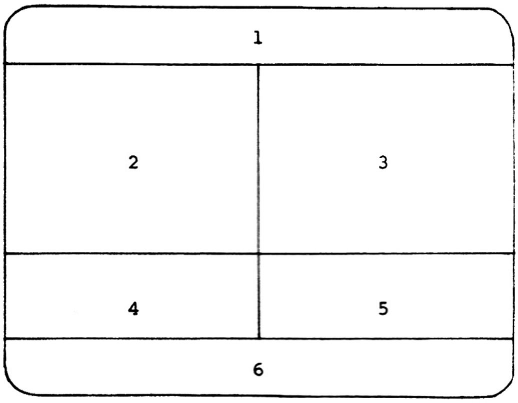
(2) 6502 Stack with Stack Pointer Highlighted.

(3) Code Disassembly and Trace/Single-Step Options.

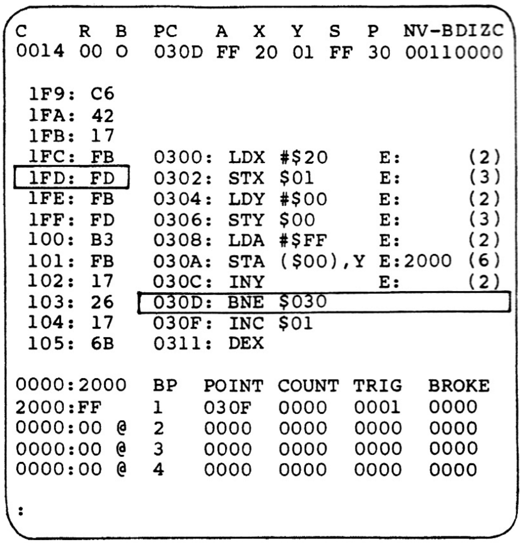
(4) User-selected Memory Cells.

(5) User-selected Breakpoints.

(6) Bugbyter Command Line.



A typical Bugbyter display looks like this:



```
C    R  B  PC   A  X  Y  S  P  NV-BDIZC
0014 00 0  030D FF 20 01 FF 30 00110000

 1F9: C6
 1FA: 42
 1FB: 17
 1FC: FB   0300: LDX #$20     E:      (2)
 1FD: FD   0302: STX $01      E:      (3)
 1FE: FB   0304: LDY #$00     E:      (2)
 1FF: FD   0306: STY $00      E:      (3)
 100: B3   0308: LDA #$FF     E:      (2)
 101: FB   030A: STA ($00),Y E:2000  (6)
 102: 17   030C: INY          E:      (2)
 103: 26   030D: BNE $030
 104: 17   030F: INC $01
 105: 6B   0311: DEX

0000:2000  BP  POINT COUNT TRIG  BROKE
2000:FF    1   030F  0000  0001  0000
0000:00 @  2   0000  0000  0000  0000
0000:00 @  3   0000  0000  0000  0000
0000:00 @  4   0000  0000  0000  0000

:
```

REGISTER SUBDISPLAY 1

In the above Master Display, Bugbyter is displaying the six 6502 registers at the top of the screen. They are as follows:

		In This Example
PC	is the Program Counter	030D
A	is the A-register	FF
X	is the X-register	20
Y	is the Y-register	01
S	is the Stack pointer	FF
P	is the Processor status	30

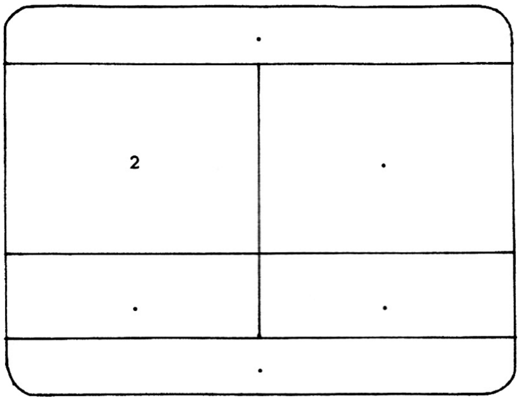
Note that in the far upper right, the processor status (P) is not only represented as a two-digit hex number, but is also broken down into its individual bits (NV-BDIZC), where:

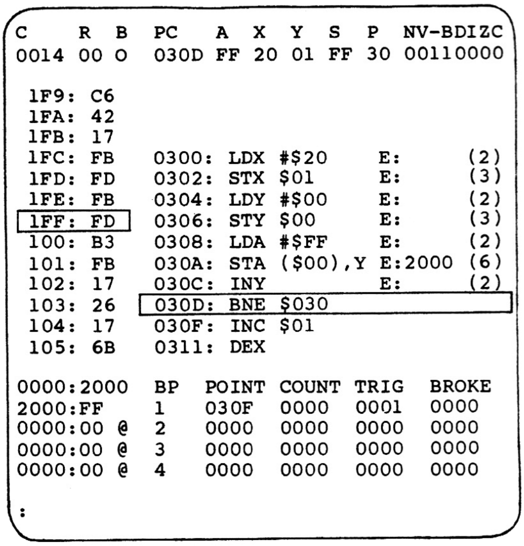
		In This Example
N	is the Negative bit	0
V	is the oVerflow bit	0
-	is unused	1
B	is the Break bit	1
D	is the Decimal bit	0
I	is the Interrupt bit	0
Z	is the Zero bit	0
C	is the Carry bit	0

The three remaining registers at the top of the screen in the register subdisplay are: C (see Option=E in Trace/Single-Step Chapter VIII), R for Trace rate (see Chapter VIII) and B for breakpoints IN or OUT (see Chapter IX).

STACK SUBDISPLAY 2

Just below and to the left of subdisplay 1 is subdisplay 2, a window into the 6502's stack. This stack subdisplay contains a column of ascending addresses and an adjacent column of corresponding contents of the stack address cells.





```
C     R  B   PC    A  X  Y  S  P  NV-BDIZC
0014 00 0   030D FF 20 01 FF 30 00110000

 1F9: C6
 1FA: 42
 1FB: 17
 1FC: FB   0300: LDX #$20     E:      (2)
 1FD: FD   0302: STX $01      E:      (3)
 1FE: FB   0304: LDY #$00     E:      (2)
 1FF: FD   0306: STY $00      E:      (3)
 100: B3   0308: LDA #$FF     E:      (2)
 101: FB   030A: STA ($00),Y E:2000  (6)
 102: 17   030C: INY          E:      (2)
 103: 26   030D: BNE $030
 104: 17   030F: INC $01
 105: 6B   0311: DEX

0000:2000  BP  POINT COUNT TRIG  BROKE
2000:FF    1   030F  0000  0001  0000
0000:00 @  2   0000  0000  0000  0000
0000:00 @  3   0000  0000  0000  0000
0000:00 @  4   0000  0000  0000  0000

:
```

Notice that one row in the stack subdisplay is in highlighted, inverse video (in the above example, 1FF: FD). This signifies the current address of the stack pointer as confirmed by the FF in the register subdisplay 1.

Setting the stack pointer. Type:

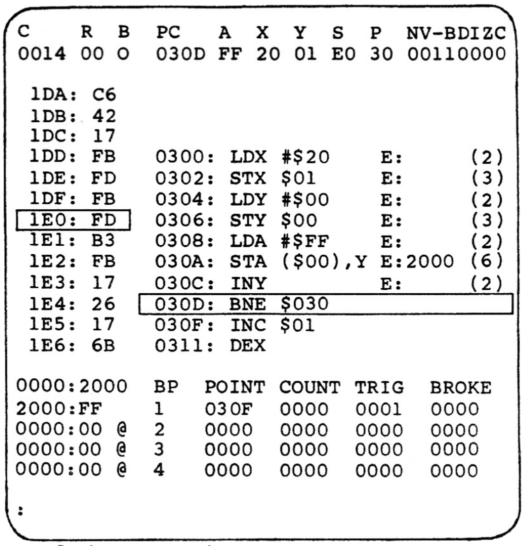
S=E0 RETURN

Three changes should occur to Bugbyter's Master Display:

(1) The command line (bottom of the screen) should display S=E0 and then clear after the RETURN key is pressed, leaving just the command prompt (:).

(2) The stack pointer value in subdisplay 1 should change to E0 (under the letter S).

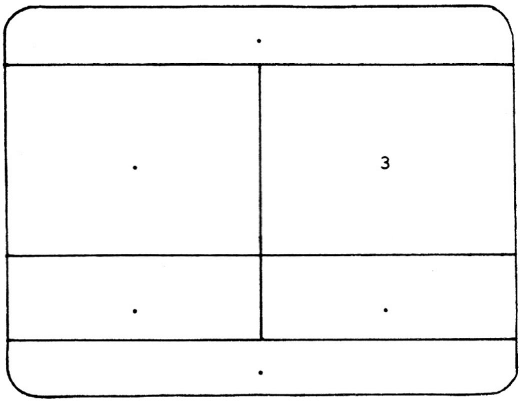
(3) The stack window should show the new stack pointer address (1E0) in the center of subdisplay 2.


```
C    R  B  PC   A  X  Y  S  P  NV-BDIZC
0014 00 0  030D FF 20 01 E0 30 00110000

 1DA: C6
 1DB: 42
 1DC: 17
 1DD: FB   0300: LDX #$20     E:      (2)
 1DE: FD   0302: STX $01      E:      (3)
 1DF: FB   0304: LDY #$00     E:      (2)
 1E0: FD   0306: STY $00      E:      (3)
 1E1: B3   0308: LDA #$FF     E:      (2)
 1E2: FB   030A: STA ($00),Y E:2000  (6)
 1E3: 17   030C: INY          E:      (2)
 1E4: 26   030D: BNE $030
 1E5: 17   030F: INC $01
 1E6: 6B   0311: DEX

0000:2000  BP  POINT COUNT TRIG  BROKE
2000:FF    1   030F  0000  0001  0000
0000:00 @  2   0000  0000  0000  0000
0000:00 @  3   0000  0000  0000  0000
0000:00 @  4   0000  0000  0000  0000

:
```

(Note: The contents of the stack in these and all the other examples in this manual will no doubt differ from those on your screen. The stack addresses, however, should match.

The stack pointer in subdisplay 1 shows the address as E0, while subdisplay 2 indicates the stack pointer as $1E0. The 6502's stack is fixed in the second page of memory from $100 to $1FF. Bugbyter displays all stack addresses as 1HH, a three-digit hex address. To adjust the stack pointer, the 6502 requires only a single byte--TXS. Bugbyter allows either format for adjusting the stack pointer. S=1E0 would have had the same effect as S=E0 in the above example.

DISASSEMBLY SUBDISPLAY 3



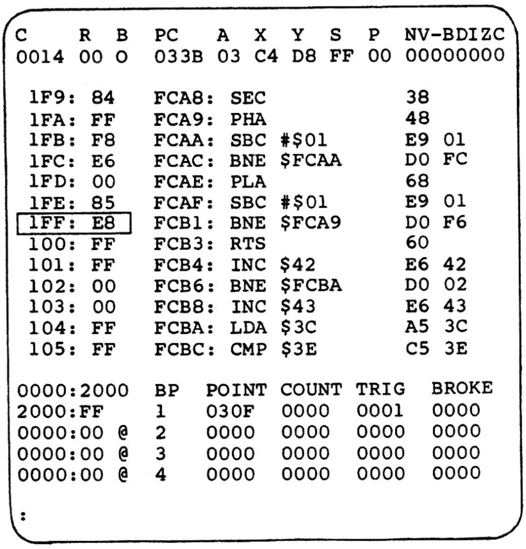
Just to the right of the stack subdisplay is the code disassembly and options subdisplay 3. Bugbyter uses this window to display the user's program code in the form:

address:opcode operand option

As an example, type:

FCA8L RETURN

Subdisplay 3 should show:



```
C    R  B   PC   A  X  Y  S  P  NV-BDIZC
0014 00 0   033B 03 C4 D8 FF 00 00000000

 1F9: 84    FCA8: SEC           38
 1FA: FF    FCA9: PHA           48
 1FB: F8    FCAA: SBC #$01      E9 01
 1FC: E6    FCAC: BNE $FCAA     D0 FC
 1FD: 00    FCAE: PLA           68
 1FE: 85    FCAF: SBC #$01      E9 01
 1FF: E8    FCB1: BNE $FCA9     D0 F6
 100: FF    FCB3: RTS           60
 101: FF    FCB4: INC $42       E6 42
 102: 00    FCB6: BNE $FCBA     D0 02
 103: 00    FCB8: INC $43       E6 43
 104: FF    FCBA: LDA $3C       A5 3C
 105: FF    FCBC: CMP $3E       C5 3E

0000:2000   BP  POINT COUNT TRIG  BROKE
2000:FF     1   030F  0000  0001  0000
0000:00 @   2   0000  0000  0000  0000
0000:00 @   3   0000  0000  0000  0000
0000:00 @   4   0000  0000  0000  0000

:
```

In this example, subdisplay 3 contains a short disassembly of a part of the Apple Monitor ROM's WAIT routine. The disassembly display format is nearly identical with Apple's Monitor disassembler.

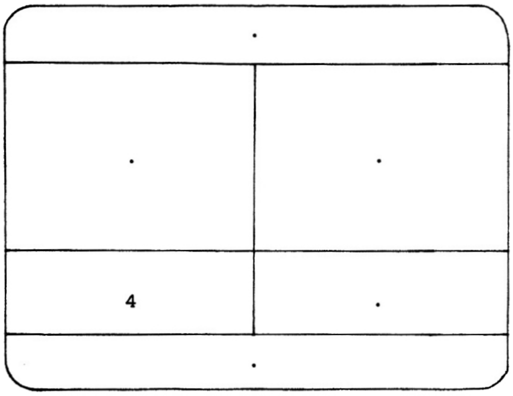
In the above example, the fourth line of the disassembly screen reads:

```
                         FCAC: BNE $FCAA     D0 FC

where:  FCAC     is the hex address
        BNE      is the 6502 opcode
        $FCAA    is the operand (an address in this case)
        D0 FC    are the actual bytes in memory $FCAC and FCAB
```

The actual bytes (D0 FC) are in the Bugbyter's option field. There is only one option (O=B) for disassemblies (L). Six more options are available in the Trace/Single-Step mode. (See Chapter VIII).

MEMORY CELL SUBDISPLAY 4

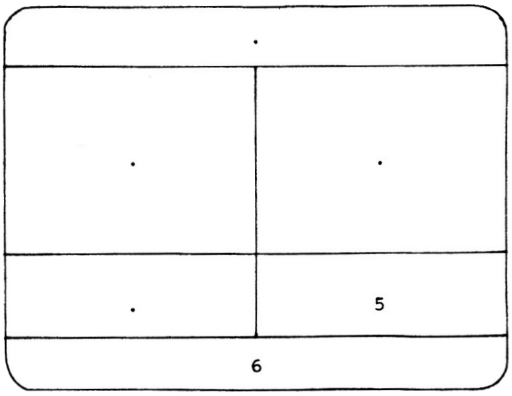
Just below the stack subdisplay 2 is the memory cell subdisplay 4. User-selected cells containing single bytes and/or byte pairs (addresses) are continually displayed in the lower left corner of the Bugbyter Master Display. See Chapter VII for a full explanation of function and use of subdisplay 4.



BREAKPOINT and COMMAND LINE SUBDISPLAYS 5 and 6

Below subdisplay 3 is the user-selected breakpoint subdisplay 5. The user may select one or more addresses that will cause Bugbyter-controlled interruption of any program being traced. Chapter IX is devoted to the use of breakpoints.

The Bugbyter command line, subdisplay 6, occupies the bottom row of the Master Display. Enter Bugbyter commands here using the mini line editor discussed in the next chapter (VI).

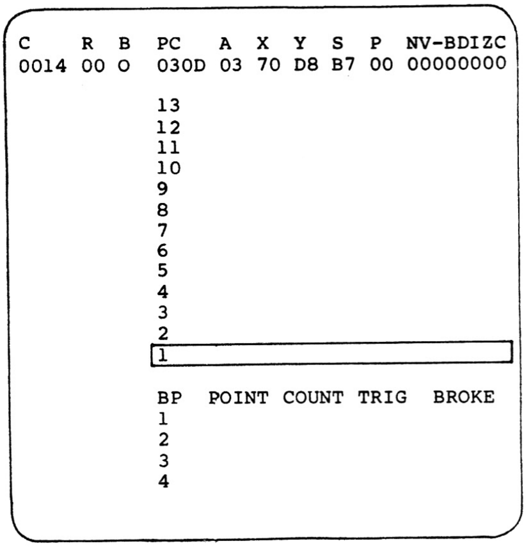


SET

Subdisplays 1 and 6 are fixed. That is, two lines of the Bugbyter Master Display are dedicated to register display (1) and one for command entry and editing (6). The SET command allows you to alter the relative size of the remaining subdisplays, 2, 3, 4 and 5, and the positions of the stack pointer and next-instruction-to-be-executed demarkation in subdisplays 2 and 3. Type:

SET RETURN

and the Bugbyter Master Display will change to:



```
C    R  B  PC   A  X  Y  S  P  NV-BDIZC
0014 00 0  030D 03 70 D8 B7 00 00000000

           13
           12
           11
           10
           9
           8
           7
           6
           5
           4
           3
           2
           1

           BP  POINT COUNT TRIG  BROKE
           1
           2
           3
           4
```

Now you can:

(1) Use the ← and → keys to increase or decrease the number of break-points which simultaneously decreases or increases the size of the disassembly subdisplay. When satisfied, press the RETURN key to move to the next subdisplay adjustment.

(2) Use the ← and → keys to move the next-instruction-to-be-executed inverse bar. The position of this bar divides the rows available in subdisplay 3 among the instructions just executed (above the bar) and instructions not yet executed (at and below the bar) for the Trace/Single-Step mode. Press RETURN when satisfied with the bar postion.

(3) Use the ← or → key to adjust the lines available for the stack (sub-display 2) versus the MEM (subdisplay 4). Press RETURN to continue.

(4) Use the ← or → key to position the stack pointer in stack subdisplay 2. Press RETURN to return to the Bugbyter Master Display.

The SET command does not affect the contents of any subdisplay except sub-display 3. Specifically, any MEM registers or breakpoints that have been assigned but are not displayed have not been lost. Using another SET command to adjust the Master Display can make them reappear.

CHAPTER VI

SELECTED COMMANDS

THE COMMAND LINE EDITOR

The command line at the bottom of the Master Display is a 40 character, horizontal scrolling window into a 128 character buffer. The following keys have the standard Apple input functions:

RETURN	accept user-entered commands from the beginning of the cursor
←	move the cursor to the left one character
→	move the cursor to the right one character
CTRL X	delete entire command line

Five additional functions are available for Bugbyter command line editting:

CTRL B	move the cursor to the beginning of the command line
CTRL N	move the cursor to the end of the command line
CTRL D	delete one character
CTRL I	enter insert character mode (pressing any other editor function key will cause you to exit from this mode).
CTRL C	accept next keystroke verbatim

The SPACE bar has one special function: If the very first character you enter on the command line is a space, Bugbyter will display the next available memory address (last used memory address plus one). This is very handy for memory reference (see Chapter VII) and the ASM command (next section).

ASM

The Bugbyter ASM command is comparable to the Apple Monitor's mini-assembler. That is, both the Apple mini-assembler and Bugbyter will accept a hex address followed by a ":" followed by a 6502 instruction. For example, type:

```
ASM  RETURN
```

and Bugbyter will clear the disassembly subdisplay and place you in ASM mode. Then type:

```
300:LDA C000  RETURN
<space> BPL 300  RETURN
```

Notice that Bugbyter automatically calculates the next available address ($303) and even prints it on the command line before you type "BPL 300". Each instruction entered will be placed at the bottom of the disassembly subdisplay and the previously entered instructions will be scrolled up.

DISASSEMBLY (L)

To disassemble a block of code, Bugbyter will accept an: address L RETURN or just L RETURN. Example:

```
FCA8L   RETURN
```

will disassemble the first few lines of the Monitor's WAIT ($FCA8) routine.

```
L   RETURN
```

will continue to disassemble a few more lines.

MONITOR (M)

To enter the Apple Monitor's command mode (*), press M RETURN. The Monitor can be used for block memory moves (see Apple Reference Manual) a feature not provided by Bugbyter. To return to Bugbyter command mode, press CTRL Y and then RETURN.

DOS COMMANDS (.)

You can enter Apple DOS commands from the Bugbyter command line by preceding them with a period. Example:

```
.CATALOG   RETURN
```

Once the catalog listing (or any DOS operation directed through Bugbyter) has been completed, press RETURN to re-enter Bugbyter.

NOTE: Any DOS errors will not leave you in Bugbyter after the error message, is printed, but instead will return you to Basic. Type:

```
CALL 1016   RETURN
```

to re-enter Bugbyter. DOS errors encountered while Bugbyter is located in the language card have more serious effects. Press the RESET key to counteract the effects, then type: CALL 1016 RETURN.

REGISTER REFERENCE

All the registers displayed at the top of the Bugbyter Master display are user assignable. For example, type:

```
A=8D   RETURN
```

and the value immediately under the A (for accumulator) on the top row will change to hexadecimal 8D. You can use C, R, PC, A, X, Y, S or P followed by an "=" followed by a hex value to assign any of these eight registers. B is not a register (see Chapter IX--"Breakpoints") and NV-BDIZC is the binary value of the P register (see Chapter V--"Register Subdisplay"). C and R are special Bugbyter registers used in Trace/Single-Step mode (see Chapter VIII).

SCREEN DISPLAY (ON/OFF)

During the tracing of a program, Bugbyter is constantly updating its Master Display. Typing:

```
OFF  RETURN
```

will cause Bugbyter to turn off subdisplays 1 thru 5, the bulk of the Master Display, leaving just the command line (subdisplay 6). This will result in a marked increase in tracing speed and will eliminate contention over screen usage with the program being traced. Typing:

```
ON  RETURN
```

will return the Master Display to the Apple's screen with all registers reflecting the most current state of the 6502.

BASE CONVERSION

Bugbyter allows simple conversions from hexadecimal to decimal:

```
$C3=  RETURN

      or

78D=  RETURN
```

and decimal to hexadecimal:

```
+43=  RETURN

      or

-15119=  RETURN
```

QUITTING (Q)

Press Q RETURN to exit Bugbyter and return to Basic+DOS.

CHAPTER VII

MEMORY REFERENCE

There are two ways to display selected memory cells:

(1) Using the memory display page to display 184 contiguous memory cells in both hexadecimal and ASCII;

(2) Using the MEM command to edit subdisplay 4.

MEMORY ASSIGNMENT

In command mode on either the Master or memory display screen, Bugbyter can accept a memory assignment of hex bytes and/or ASCII characters. Example:

```
805: "HELLO" 8D
```

will assign the ASCII character string HELLO with the most significant bit on ("), followed by a hex 8D to memory cells 805 to 80A. Another Example:

```
2500: F 'C' 0 A3
```

will assign hex 0F to address $2500, ASCII character C with the significant bit off (') to address $2501, $00 to $2502 and $A3 to address $2503. Note that Bugbyter allows free mixture of hex and ASCII (most signficant bit on and off) in a memory assignment command.

MEM

Use the SET command if necessary to increase or decrease the size of the MEM subdisplay 4. Then type:

```
MEM  RETURN
```

which moves the cursor to the upper left corner of the MEM subdisplay. Bugbyter will now accept any one to four digit hexadecimal address. Use the → and ← keys to move to the next or previous address. Preceding an address, you have the option to type:

H to display the contents of the address as hex and ASCII, or

P to display the contents of the address and address+1 as a pointer (most significant byte first).

To exit the MEM subdisplay, press the ESC key.

MEMORY DISPLAY PAGE

To display a screen's worth of hex and ASCII, type: address: RETURN. Example:

AA60: RETURN

will cause Bugbyter to switch from the Master Display to a memory display with $AA60 as the first address in the upper left corner.

```
AA60:  2D 00 00 00 02 00 00 00  -
AA68:  01 00 06 00 00 00 00 00
AA70:  00 00 00 03 00 A0 A0 A0
AA78:  A0 A0 A0 A0 A0 A0 A0 A0
AA80:  A0 A0 A0 A0 A0 A0 A0 A0
AA88:  A0 A0 A0 A0 A0 A0 A0 A0
AA90:  A0 A0 A0 A0 A0 A0 A0 A0
AA98:  A0 A0 A0 A0 A0 A0 A0 A0
AAA0:  A0 A0 A0 A0 A0 A0 A0 A0
AAA8:  A0 A0 A0 A0 A0 A0 A0 A0
AAB0:  A0 03 84 00 00 00 00 00
AAB8:  C1 D0 D0 CC C5 D3 CF C6  APPLESOF
AAC0:  D4 E8 B7 BB B3 BB B4 00  Th7;3;4
AAC8:  C0 7E B3 21 AB 05 AC 57  @~3!+E,W
AAD0:  AC 6F AC 2A AD 97 AD EE  ,o,*-W-n
AAD8:  AC F5 AC 39 AC 11 AD 8D  ,u,9,Q-
AAE0:  AE 17 AD 7E B3 7E B3 89  .W-~3~3
AAE8:  AC 95 AC 86 AC 92 AC 7E  ,U,F,R,~
AAF0:  B3 7E B3 BD AC C9 AC BA  3~3=,I,:
AAF8:  AC C6 AC 7E B3 E0 00 F0  ,F,~3`@p
AB00:  02 A2 02 8E 5F AA BA 8E  B"BN_*:
AB08:  9B B3 20 6A AE AD BB B5  [3 j.-;5
AB10:  C9 0D B0 0B 0A AA BD CA  IM0KJ*=J
:
```

The cells are displayed in a manner similar to that of the Apple Monitor, that is, eight cells to the line with the beginning address to the left of the cells. Bugbyter, however, adds an extra feature. To the right of each row of cells are the eight Apple ASCII characters corresponding to each of the eight hex cells. Apple ASCII is:

00-3F	inverse characters
40-7F	flashing characters
80-FF	normal characters (two sets of alphabetic characters)

The command line is still at the bottom of the screen and can accept another "address:" or memory assignment or the ESC key to return to the Master Display.

CHAPTER VIII

TRACE/SINGLE-STEP MODE

Bugbyter's Trace/Single-Step mode with its seven disassembly options and seventeen single-keystroke commands, represents a powerful debugging environment. Bugbyter is capable of tracing practically any executable 6502 program, including interrupt and timing-sensitive code. In general, the Trace/Single-Step mode is simple to use, as exemplified by the tutorial in Chapter III. But it also offers a variety of options and single keystroke commands that vastly expand the Bugbyter's capabilities. This chapter will cover these options and commands in Trace/Single-Step mode. The following chapter will introduce the use of Breakpoints, an extension of the Trace/Single-Step mode, for selective interruption of the program being traced.

OPTIONS

During Trace/Single-Step operation, on the right side of the disassembly subdisplay, a user may select one of the following display options. In Bugbyter command mode, BEFORE entering Trace/Single-Step mode, typing:

O=A	RETURN	will display the 6502 accumulator in binary
O=X	RETURN	will display the 6502 X-register in binary
O=Y	RETURN	will display the 6502 Y-register in binary
O=S	RETURN	will display the 6502 Stack pointer in binary
O=P	RETURN	will display the 6502 Processor status in binary
O=B	RETURN	will display the instruction bytes in hex
O=E	RETURN	will display computed effective addresses, relative branches and instruction cycles.

The last option, O=E, is probably the most powerful of all the options and requires some extra discussion. There are four 6502 addressing modes for which the 6502 internally computes an effective address. They are:

mode	example
indexed	LDA $300,X
indirect	JMP ($300)
indexed indirect	LDA ($10,X)
indirect indexed	LDA ($10),Y

The actual or effective address is computed based on the current contents of registers or memory cells at the time of execution. During simulated execution (Trace/Single-Step mode), with O=E set, Bugbyter will compute these effective addresses and display them in the disassembly subdisplay. Also the E option will display all relative branches (the hex byte operand of a branch opcode).

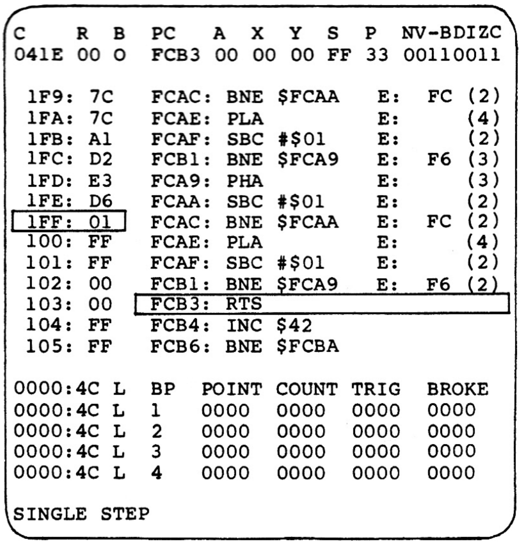
At the far right of the disassembly subdisplay during Trace/Single-Step with the E option set, are cycle counts, shown in parentheses, for each instruction executed. For example, type:

C=0 RETURN	(clears the Bugbyter cycle counter)
O=E RETURN	(sets the Trace/Single-Step Option to E)
A=12 RETURN	(sets the accumulator to $12)
FCA8S RETURN	(executes the first instruction in Monitor WAIT routine)
R	(is the Trace/Single-Step command--trace until RTS)

The Bugbyter Master Display will begin tracing the Monitor WAIT routine. The command line, the bottom of the Master Display, will show:

```
                    TRACE                         AWAITING RTS
```

and the rest of the screen will be changing rapidly. After just a few seconds, the screen will stop changing and look like the following:



```
C    R  B  PC   A  X  Y  S  P  NV-BDIZC
041E 00 0  FCB3 00 00 00 FF 33 00110011

 1F9: 7C   FCAC: BNE $FCAA   E:  FC (2)
 1FA: 7C   FCAE: PLA         E:     (4)
 1FB: A1   FCAF: SBC #$01    E:     (2)
 1FC: D2   FCB1: BNE $FCA9   E:  F6 (3)
 1FD: E3   FCA9: PHA         E:     (3)
 1FE: D6   FCAA: SBC #$01    E:     (2)
 1FF: 01   FCAC: BNE $FCAA   E:  FC (2)
 100: FF   FCAE: PLA         E:     (4)
 101: FF   FCAF: SBC #$01    E:     (2)
 102: 00   FCB1: BNE $FCA9   E:  F6 (2)
 103: 00   FCB3: RTS
 104: FF   FCB4: INC $42
 105: FF   FCB6: BNE $FCBA

0000:4C L  BP  POINT COUNT TRIG  BROKE
0000:4C L  1   0000  0000  0000  0000
0000:4C L  2   0000  0000  0000  0000
0000:4C L  3   0000  0000  0000  0000
0000:4C L  4   0000  0000  0000  0000

SINGLE STEP
```

The cycle counter register in the upper left corner will show $41E (=1054 decimal) CPU cycles or approximately .001 seconds to execute the WAIT routine when the accumulator is preset to $12. The cycle register will only count during Trace/Single-Step mode and when option E is set.

Note: The cycle counter will not count when the Bugbyter Master Display is OFF.

Notice that the command line prompt (:) disappears when in Trace/Single-Step mode. This signifies that the standard Bugbyter commands are not available and only a new set of single keystroke, Trace/Single-Step commands will be accepted. They are:

SPACE	Single step one opcode.
RETURN	Continuous trace.
ESC	Return to Bugbyter command line.
R	Trace until RTS opcode encountered.
→	Skip next instruction.
C	Clear Cycle Counter.
P	Use Paddle 0 to adjust Trace Rate.
K	Use Keyboard Rate (R=value) to adjust Trace Rate.
Q	Sound off (Quiet).
S	Sound on.
1	Display primary Apple screen.
2	Display secondary Apple screen.
T	Display Apple Text screen.
L	Display Apple Lores graphics screen.
H	Display Apple Hires graphics screen.
F	Display Full screen graphics.
M	Display Mixed text and graphics.

All these keys need only one stroke to operate. Use the ESC key to exit from Trace/Single-Step mode and return to the Bugbyter Master Display.

Trace/Single-Step mode may be re-entered and program code continued at any time by typing: S RETURN or T RETURN. Bugbyter will retain the Trace/Single-Step continuation address even when other Bugbyter commands and operations are interspersed.

RATE ADJUSTMENT

During tracing, Bugbyter is interpreting each 6502 instruction of your program. That is, the Apple CPU is executing the Bugbyter program, which in turn is executing your program code. The obvious and visible result is that code being traced with Bugbyter will execute slower than if it were executed directly by the 6502 microprocessor. The rate of tracing can be adjusted in the following ways:

(1) Before Trace/Single-Step mode is entered, type R= followed by a hex value from 0 to FF; where 0 is the fastest rate (default) and FF is the slowest. Then press RETURN.

(2) During Trace/Single-Step mode, press the P key and use paddle 0 to adjust the rate. (Pressing the K key will disable the paddle and return to the keyboard entered rate).

(3) Before Trace/Single-Step mode is entered, type OFF and press RETURN to disable the Bugbyter Master Display. This will greatly increase the speed of tracing. (Type ON and RETURN after exiting from Trace/ Single-Step mode to restore the Master Display).

CAUTION — MEMORY CONTENTION

Bugbyter is a self-contained 6502 program that needs memory on the zero page and the stack. Bugbyter saves to, then restores from internal memory, all zero page cells it will use. The effect is NO zero page impact on code being debugged by Bugbyter!

The same is not true for the stack. Stack locations $100 to 11F (the first 32 decimal cells of the 6502 stack) are reserved for Bugbyter. If the stack pointer is set to any address between $00 to $1F, Bugbyter will alert you by flashing the ends of the stack pointer's inverse bar in the stack subdisplay. Try to avoid using the beginning of the stack.

CAUTION — REAL-TIME CODE

Some code that you can trace may require execution in native 6502 mode, that is, it may need to execute at the full speed of the Apple's CPU. Tracing it at any rate slower that 1 MHz per machine cycle will cause it to function incorrectly or not at all. A prime example is the core routines associated with the Apple's DOS. The read data, write data, read address and track seek routines are very sensitive to cycle speed variation. These core routines will not function at all if traced. Bugbyter does offer a solution, a method of allowing subroutines to execute in native mode while tracing the outer levels. See Chapter XI for a complete discussion of debugging real-time code.

CAUTION — SCREEN CONTENTION

Many programs that you may trace will direct output to the same screen Bugbyter uses--text screen page one. This is especially noticeable when your program calls the Apple Monitor's scroll routine. To restore the Bugbyter screen, you can type:

```
ON   RETURN
```

CHAPTER IX

BREAKPOINTS

The Trace/Single-Step mode is described in the previous chapter and understanding how to use it is a prerequisite of this chapter.

Breakpoints provide a means of selectively interrupting program execution. Bugbyter offers two types of breakpoints, transparent and real. Both types are monitored and managed by Bugbyter and are discussed in this chapter.

BREAKPOINT SUBDISPLAY

In the lower right area of the Master Display is the breakpoint subdisplay. Use the SET command to increase or decrease the number of available breakpoints. The breakpoint subdisplay has four field column headings:

POINT	is the user-defined breakpoint address.
COUNT	is the number of times the POINT address has been encountered.
TRIG	is the user-defined count before breaking.
BROKE	is the number of times Bugbyter has been TRIGgered.

To enter a breakpoint address, type "BP" followed by the breakpoint row number. Example:

```
BP1  RETURN
```

Bugbyter will move the cursor to the first zero in the POINT field. Enter a hexadecimal number for the address of breakpoint 1. Use the arrow keys to move from field to field in breakpoint 1. Move to the TRIG field and assign it a hex value greater than zero. (TRIG set to 0 will cause Bugbyter to ignore breakpoint 1). During Trace/Single-Step mode, trans- parent or real breakpoints are monitored such that every time a breakpoint address (POINT) is encountered, the COUNT value is incremented and compared to the user-set TRIG value for that breakpoint. When the COUNT equals the TRIG, Bugbyter stops Trace mode BEFORE executing the instruction at the POINT address, inverses the breakpoint row that caused the break in the breakpoint subdisplay and exits from Trace/Single-Step mode. (It also clears the COUNT). The user may continue tracing, that is, re-enter the Trace/Single-Step mode, by pressing T RETURN or S RETURN.

To clear a breakpoint, type: CLR followed by the breakpoing number and RETURN. To clear all breakpoints, type: CLR RETURN.

TRANSPARENT BREAKPOINTS

The Bugbyter default method of monitoring breakpoints during tracing is interpretive, that is, transparent. During Trace/Single-Step, operation Bugbyter is monitoring the program counter (PC) and directly comparing the PC to the POINTS set up in the breakpoint subdisplay. 6502 break opcodes (00) are not installed (OUT) and therefore, do not cause program interruption. (Break opcodes could of course exist in the original code being traced and would cause Bugbyter simply to exit Trace/Single-Step mode).

From Bugbyter command mode (:), typing:

```
OUT   RETURN
```

will force Bugbyter to transparent mode, break opcodes OUT. In the register subdisplay at the top of the screen, under the B, should be the letter "O".

REAL BREAKPOINTS

From Bugbyter command mode, type:

```
IN   RETURN
```

In the register subdisplay under the B, should now appear an "I". Bugbyter will now install 6502 break opcodes (00) at all user-assigned breakpoints (POINT addresses with their associated TRIG's set to greater than zero).

While tracing, Bugbyter will still monitor the program counter (PC) and interrupt the Trace/Single-Step mode. Bugbyter also is capable of allowing the 6502 to execute the program directly. Any break opcodes installed by the IN command will return control back to Bugbyter, inversing the breakpoint row that contains the POINT address in the breakpoint subdisplay and entering Bugbyter command mode (just as with transparent breakpoints).

Two Bugbyter commands are available for initiating direct code execution: From command mode, type: starting address G RETURN or starting address J RETURN. Example:

```
300G   RETURN
1A1FJ   RETURN
```

If the starting address is left out of the command, Bugbyter will use either the last trace address of the last starting address specified (whichever has most recently been entered). The G command is similar to the Apple Monitor's G command; a return from subroutine (RTS) will return to Bugbyter. Since a J command does not push a return address on the stack, an RTS will use an undefined address from the stack if encountered. When first executing your code, type: starting address G RETURN. After encountering any break opcodes, type:

```
J   RETURN
```

to continue direct, real-time execution.

CAUTION: Once IN has been set, Bugbyter has altered your program by inserting break opcodes at every POINT address. If you exit Bugbyter before typing OUT, your code may be riddled with unwanted 6502 breaks. Be sure to type:

```
OUT   RETURN
```

to return your code to its original condition.

Also, in the IN mode, Bugbyter will not allow you to add, clear or edit any breakpoints. Issuing the OUT command is necessary for any breakpoint modification.

CHAPTER X

SOFT SWITCHES

A group of "soft switches" are located near the beginning of the Bugbyter program code. They are used to control some miscellaneous functions described in this chapter. The heading "relative location" means the address in RAM, offset from the beginning of Bugbyter. "Absolute location" assumes the default starting address of $7C00. If you BRUN or BLOAD or use BUGLOADER to relocate Bugbyter to another starting address, you will have to adjust the "absolute location" accordingly.

UNDEFINED OPCODES

Relative Location	Absolute Location	Function
start +3	7C03	execute undefined opcodes (default=OFF)

During Trace/Single-Step, Bugbyter will ignore illegal, undefined 6502 opcodes, when start +3 ($7C03) is set to 0. If start +3 is set to $80, Bugbyter will execute all undefined opcodes. This is useful for exploring undefined operations of the 6502. (Try tracing AF 58 FF). Since Bugbyter does not know the length of the undefined opcode's operand, Trace/Single-Step will assume no operand and just increment the PC by one. It's up to you to SKIP (-> during Trace/Single-Step) past the operand, if any, to the next opcode. Using Bugbyter with its complete register and memory display will allow you to map all the undefined 6502 opcodes.

The following three soft switches can be set so as to remove the contention of Bugbyter and your program over use of the paddle button, paddle and keyboard.

PADDLE BUTTON 0

start +4	7C04	use paddle button 0 for Trace suspend (default=OFF)

Setting 7C04 to $80 (7C04:80 in command mode) will allow the use of paddle button 0 to suspend tracing. Caution: If the paddles are not connected to the Apple's Game I/O port, Trace will freeze your Apple -- disconnected paddles are equivalent to continuously pressing the buttons with them connected.

PADDLE 0

start +5	7C05	use paddle 0 for Trace rate adjustment (default=OFF)

The Trace rate can be preset by keyboard input (R=value) or by pressing P and transferring control to paddle 0 in Trace/Single-Step mode. Setting $7C05 to 0 (7C05:0), will cause Bugbyter to ignore a user pressed P key during Trace/Single-Step operation.

KEYBOARD

start +6	7C06	Trace/Single-Step keyboard polling (default=ON)

During tracing, Bugbyter is sampling (polling) the Apple keyboard for any of the Trace/Single-Step mode single keystroke commands. Therefore, a program that is being traced that expects input from the keyboard will never get a character unless Bugbyter's polling is disabled. Setting $7C06 to any hex value with the most significant bit on will allow the program being traced to accept all characters from the keyboard except one--the Apple ASCII character that is set in $7C06. For example, if $7C06 is set to $81 before entering Trace/Single-Step mode, Bugbyter will pass any and all characters arriving from the keyboard except CTRL A ($81). Pressing CTRL A will cause Bugbyter to stop tracing and return to command mode. This is useful for tracing software that requires input from the keyboard like Integer Basic.

SOUND

start +7	7C07	Sound switch (default=ON)

During Trace/Single-Step operation , pressing the Q key will turn off the clicks and S will turn the clicks back on. The most significant bit of $7C07 is affected directly by pressing Q or S during Trace/Single-Step mode or by manually setting $7C07 in command mode.

CYCLE COUNTER

start +8,+9	7C08,7C09	Cycle counter

During Trace/Single-Step operation with option E set (O=E), Bugbyter will update the cycle counter displayed in the upper left corner of the Master Display. $7C08,7C09 contain the low,high bytes of the C register.

Start +A to start +D ($7C0A to $7C0D), the last soft switches, are discussed in the next chapter.

REAL-TIME EXECUTION

The last pair of soft switches (see previous chapter) allows for user specification of a region of code that will execute in native 6502 mode, that is, at the full speed of the Apple's CPU. Offset from the beginning of the Bugbyter program by +$A and +$B is the user-definable starting address of the region, and at offset +$C and +$D is the region's ending address. Any subroutine calls (JSR's) to inside that specified region will cause Trace/Single-Step mode to transfer full control over to the 6502 CPU. When a return from subroutine (RTS) is encountered, the 6502 CPU will re-enter Bugbyter Trace/Single-Step mode. Example: With 48K DOS in RAM, typing:

```
     7C0A:0 B8 FF BF    RETURN        (real-time starting address=$B800,
                                      ending address=$BFFF)

     300:JSR A56E   RETURN            (DOS Catalog routine)

     303:BRK   RETURN                 (code termination)

     OFF   RETURN                     (Master Display off)

     300T   RETURN                    (start tracing)
```

will cause Bugbyter to begin tracing the Apple DOS's Catalog routine until there is a JSR to the Read-Write-Track-Sector (RWTS) routine at $BD00 -- inside our real-time address region ($B800-BFFF). At that time, Bugbyter allows RWTS to execute directly under the 6502 CPU -- seeking tracks and reading sectors from the diskette in real-time. When the RWTS routine exits back to DOS (RTS), Bugbyter is re-entered and the code that followed the call (JSR) to RWTS is again traced under the Trace/Single-Step mode.

Note that the previous example executed fairly slowly. The reason was that the Monitor ($F800 to FFFF) was outside the specified real-time region and character output, especially text scrolling, was executing in Trace/ Single-Step mode, that is, slowly. To increase the execution speed, increase the real-time region's range to include the Monitor. Type:

```
     7C0C:FF FF   RETURN              (ending address = $FFFF)
```

Then type:

```
     300T   RETURN
```

Notice the increase in speed. At any time, you can press the ESC key to return to Bugbyter command mode. Also you can type:

```
     ON   RETURN
```

to resurrect the Master Display.

APPENDIX A

NOTE: All addresses and values are in hex unless stated otherwise.

The ESC key is generally used to return to the command line.

COMMAND LINE EDITOR

NOTE: The command line is a 40 character window into a 128 character buffer at the bottom of the Bugbyter master display.

Key	Function
RETURN	Accept user-entered command line.
SPACE	If the first character, display next-available memory address to be filled (used for memory reference and ASM command).
←	Move cursor to the left.
→	Move cursor to the right.
CTRL B	Move cursor to beginning of command line.
CTRL C	Accept next keystroke verbatim.
CTRL D	Delete a character.
CTRL I	Enter insert character mode (any other editor function exits from this mode).
CTRL N	Move cursor to end of command line.
CTRL X	Delete command line.

GENERAL COMMANDS

ASM	Enter assembler mode: Clear disassembly subdisplay and display user-entered 6502 mnemonics--compatible with Apple mini-assembler.

addressL L	Disassemble code beginning at address (addressL) or continue disassembling (L).
M	Enter Apple Monitor. Return to Bugbyter with CTRL Y.
SET	Customize master Bugbyter display where:
	→ Moves window down.
	← Moves window up.
	RETURN Fixes subdisplay and advances to next subdisplay.
ON	Turn Bugbyter master display on.
OFF	Turn Bugbyter master display off.
.doscommand	Execute DOS command. Press RETURN to return to Bugbyter.
+decimalvalue=	Convert positive decimal to hex.
-decimalvalue=	Convert negative decimal to hex (65536-decimalvalue).
value=	Convert hex to decimal.
$value=	Convert hex to decimal.
V	Display copyright and version number.
Q	Quit Bugbyter (exits thru DOS vector $3D0).

REGISTER REFERENCE

PC=address	Set 6502 Program Counter with hex address.
A=value	Set 6502 A-register with hex value.
X=value	Set 6502 X-register with hex value.
Y=value	Set 6502 Y-register with hex value.
S=value	Set 6502 Stack pointer with hex value.
P=value	Set 6502 Processor Status register with hex value.
C=value	Set Bugbyter Cycle Counter to value.
R=value	Set Bugbyter keyboard Trace Rate to value.

EXECUTION COMMANDS

addressG G	Execute code as subroutine at address (addressG) or continue (G). An RTS returns to Bugbyter.
addressJ J	Jump to code at address (addressJ) or continue (J). Used with breakpoints.
addressT T	Enter Trace/Single-step mode starting at address (addressT) or continue (T). See Trace/Single-step commands.
addressS S	Enter Trace/Single-step mode and execute single opcode starting at address (addressS) or continue (S). See Trace/Single-step commands.

MEMORY REFERENCE

address:	Display 184 memory cells starting at address in hex and ASCII. Use SPACE: to display next 184 cells. Press ESC to return to Bugbyter Master display.
address:opcode or	Assign opcode mnemonic starting at address.
address:value or	Fill address with hex value.
address:"text" or	Fill address with ASCII character (MSB on).
address:'text' or	Fill address with ASCII character (MSB off).
(any mixture)	Multiple values and ASCII text (MSB on or off) can be mixed freely in memory fill. Slash (/) accepts the next character verbatim.
MEM	Edit memory subdisplay where:

H P	Display contents of address as hex and ASCII, or Display contents of address & address+1 as pointer.
address	Enter hex address of memory cell(s) to be displayed.
→ or SPACE or RETURN	Advance to next cell.
←	Return to previous cell.
ESC	Return to Bugbyter command line.

DISASSEMBLY OPTIONS FOR TRACE/SINGLE-STEP

O=A	Display 6502 Accumulator in binary.
O=X	Display 6502 X-register in binary.
O=Y	Display 6502 Y-register in binary.
O=S	Display 6502 Stack pointer in binary.
O=P	Display 6502 Processor Status register in binary.
O=B	Display instruction bytes in hex.
O=E	Display computed effective addresses or relative branches and instruction cycles.

TRACE/SINGLE-STEP

Once in Trace/Single-step mode (see T or S commands above), Bugbyter will respond to the following single keystroke commands:

SPACE	Single step one opcode.
RETURN	Continuous trace.
ESC	Return to Bugbyter command line.
R	Trace until RTS opcode encountered.
→	Skip next instruction.
C	Clear Cycle Counter.
P	Use Paddle 0 to adjust Trace Rate.
K	Use Keyboard Rate (R=value) to adjust Trace Rate.
Q	Sound off (Quiet).
S	Sound on.
1	Display primary Apple screen.
2	Display secondary Apple screen.
T	Display Apple Text screen.
L	Display Apple Lores graphics screen.
H	Display Apple Hires graphics screen.
F	Display Full screen graphics.
M	Display Mixed text and graphics.

BREAKPOINTS

BPn	Set breakpoint "n" where: value — Sets breakpoint field to value. ← — Moves to previous field → or SPACE — Moves to next field. ESC or RETURN — Returns to Bugbyter command line. POINT is a user-defined breakpoint address. COUNT is the number of times the breakpoint address was encountered. TRIG is the user-defined count before breaking. NOTE: To cause a break, TRIG must be set to one or greater. BROKE is the number of times Bugbyter triggered.
IN	Insert BRK (00) opcodes into addresses specified in breakpoint subdisplay. Disables breakpoint modification. (Used for real-time debugging).
OUT	Replace BRK opcodes with original instructions at addresses specified in breakpoint subdisplay. Enables breakpoint modification. (Used for interpretive debugging--default).
CLR	Clear all breakpoints.
CLRn	Clear breakpoint "n".

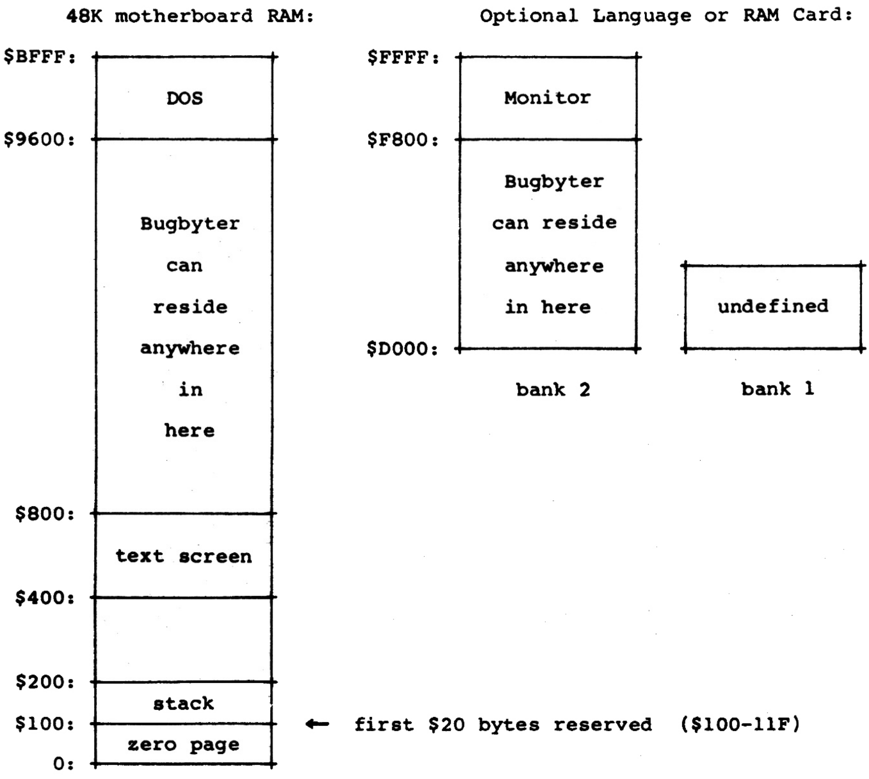
USER SOFT SWITCHES

Location	Function	
start+3	Execute undefined 6502 opcodes	($80=on, 00=off {default}).
start+4	Use button 0 for Trace suspend	($80=on, 00=off {default}).
start+5	Use paddle 0 for Trace Rate	($80=on {default}, 00=off).
start+6	Trace/Single-step keyboard polling (MSB on + ASCII character code for escape character, MSB off=normal polling {default}).	
start+7	Sound	($80=on {default}, 00=off).
start+8,+9	Cycle Counter	(low, high).
start+$A,+$B	Beginning address of real-time code	(default=$FFFF).
start+$C,+$D	Ending address of real-time code	(default=$FFFF).

APPENDIX B

TECHNICAL SPECIFICATIONS

```
BUGBYTER: -is a debugging tool for software executing under the 6502
          -written in 6502 machine language
          -has the default starting address: $7C00  (see memory map below)
          -is $1A00 bytes (6.7K) in length
          -contains self-modifying relocator
          -is relocatable from $800 to $A600 or to a language or RAM card
                  ($D000 to $DE00 -- see memory map below)
          -requires Apple II or Apple II+ or II/E
                  *with disk drive
                  *with or without game paddles
          -is compatible with Apple ///
          -is distributed on a 13-sector format diskette that is bootable
                  on a 13- or 16-sector disk controller
          -reserves the first 32 decimal bytes of the stack ($100-11F)
```


48K motherboard RAM:
$BFFF:
DOS
$9600:
Bugbyter can reside anywhere in here
$800:
text screen
$400:
$200:
stack
$100:
zero page
0:
Optional Language or RAM Card:
$FFFF:
Monitor
$F800:
Bugbyter can reside anywhere in here
$D000:
undefined
bank 2
bank 1
first $20 bytes reserved ($100-11F)

INDEX

Bugbyter is THE complete 6502 screen-oriented debugging tool. Bugbyter is a relocatable mnemonic debugger; it features a user-definable display, literal and transparent breakpoints, a resident assembler and disassembler, and is compatible with Apple DOS.

- Displays all registers
- Offers full hex and ASCII I/O
- Provides multiple options while in trace mode
- Offers RAM screen dump in hex and ASCII
- Allows single keystroke operation
- Furnishes instruction cycle counter
- Gives hexadecimal/decimal conversions
- Can run in add-on RAM card
- Is accompanied by comprehensive documentation

Computer-Advanced Ideas, Inc. is a major publisher of quality educational and technical products. CAI software is recognized for excellence of design, superior graphics, clear instructions and ease of use. As a leader in its field CAI guarantees all of its products.

Computer-Advanced Ideas Inc.
1442A Walnut Street, Suite 341
Berkeley, CA 94709