a reference manual for

B U G / 6 5

an Assebly Language Debugging program for
use with 6502-based computers built by
Apple Computer, Inc., and Atari, Inc.

PREFACE

BUG/65 is an interactive debugging tool for use in the development of assembly language programs for the ATARI 800 or ATARI 400 personal computers. It's designed to take as much of the drudgery out of assembly language debugging as possible. The design philosophy behind BUG/65 is that the computer should serve as a tool in the debugging process as opposed to a hindrance. One result of this philosophy is that BUG/65 requires a relatively large amount of memory when compared to simpler debug monitors. This is the result of a tradeoff between memory and functionality, with function winning out.

BUG/65 is a RAM loaded machine language program occupying 8K of memory; it is self relocatable as shipped and requires a full 48K bytes of memory. BUG/65 is also designed to be floppy disk based - it isn't intended to be used in cassette-only systems. BUG/65 was designed for use by an experienced assembly language programmer.

BUG/65 is an original product of the McStuff Company, which developed the product under the name "McBUG", which name is their trademark.

For use on the ATARI 800 or 400 computer with a minimum of 48K of RAM and one floppy disk drive.

TRADEMARKS

The following trademarked names are used in various places within this manual, and credit is hereby given:

OS/A+, BUG/65, MAC/65, and C/65 are trademarks of Optimized Systems Software, Inc.

Apple, Apple II, and Apple Computer(s) are trademarks of Apple Computer, Inc., Cupertino, CA

Atari, Atari 400, Atari 800, Atari Home Computers, and Atari 850 Interface Module are trademarks of Atari, Inc., Sunnyvale, CA.

SUMMARY OF MAJOR FEATURES OF BUG/65

* A full set of debugging commands - change memory, display memory, goto user program with break points, etc.

* Binary file read and write, including appended write.

* A disassembler.

* An instant assembler providing labeling capability.

* Expanded command addressing capability: hex or decimal addresses, + and - operators supported, relocated addresses supported.

* Read or write disk sector(s).

* Multiple commands permitted in a command line. Command lines can be repeated with a single keystroke or repeated forever with the special slash operator.

* Support for relocatable assemblers - the base of a module can be specified and then used to reference addresses in that module.

* BUG/65 commands can be executed from a command file, and there is a command to create command files.

* Hex to decimal and decimal to hex conversions provided.

* Memory protection of BUG/65's code and data. BUG/65 won't allow you to use a BUG/65 command that will destroy any part of BUG/65 itself. For example, you can't use the Fill command to overwrite BUG/65's code.

* Page zero sharing. BUG/65 shares page zero with a user program by keeping two copies of the shared page zero locations - one for the user and one for BUG/65 itself.

SECTION 1 : COMMAND SUMMARY

This section is intended to be a handy reference guide and will probably prove indispensable after the user has thoroughly read through the rest of this manual. For the experienced debug user, might we suggest at least a quick perusal of Sections 2 through 6 and Sections 8 and 9.

The following table is simply a syntax summary of the available commands. Excepting for the first three commands (which are described in Section 8), all the commands are described in alphabetical order in Section 7.

COMMAND CODE	SYNTAX	PURPOSE
{RETURN}		Repeat last command line
/		When appended to a command line: repeat line forever.
=		Display last command line
A	A <addr>Þ	Ascii mode memory change
B	B <addr>	Base address for relocation
C	C <startaddr1> <endaddr1> <startaddr2>	Compare memory blocks
D	D <startaddr> [<endaddr>]	Display memory
E	E #filespec	Execute a command file
F	F <startaddr> <endaddr> [<value>]	Fill memory block with value
G	G [<startaddr>] [@<breakpoint> [Rn=<value>] [I=<count>]]	Go at address, set optional breakpoint, with optional Register value breakpoint and pass Counter.
H	H <number1> <number2>	Hexadecimal arithmetic result
I	I	disk Inventory (directory listing)
J	J #filespec,string	create command file
K	K <number>	convert hex to decimal

L	L <startaddr> <endaddr> <byte1> [<byteN> ...]	Locate byte string in memory block
M	M <startaddr> <endaddr> <toaddr>	Move memory block
P	P [S] [P]	Print output on Screen and/or Printer
Q	Q	Quit...go to OS/A+
R	R [<offset>] #filespec	Read a binary file to memory with optional offset
R%	R% [<sectornumber> [<bufferaddr> [<numsectors>]]]	Read sector(s) from disk to memory buffer
S	S <addr>Þ	Substitute memory, numeric mode
T	T [S] [<count>]	Trace, with optional Skip over subroutine calls, for (optional) count intstructions.
U	U <addr> [<param>]	call User routine at given address and pass optional parameter in X,Y registers
V	V	View user registers
W	W [:A] <startaddr> <endaddr> #filespec	Write a block of memory to a binary image file, optionally appending instead of creating new file.
W%	W% [<sectornumber> [<bufferaddr> [<numsectors>]]]	Write sectors from memory buffer to disk
X	XA or XX or XY or XS or XP or XF	change user register value
Y	Y <startaddr> [<endaddr>]	dissasemble memory block
Z	Z <addr>Þ	instant assembler (at address)

SECTION 2: Notations Used In This Manual

The following notations are used in this manual:

<...> Is used to indicate a numerical address parameter. The address expression between the two characters "<" and ">" may be any valid address as described in Section 3. For example, <START> means that you can enter any valid address expression to specify the START parameter.

ß Is used to indicate one and only one blank. In most cases, blanks are insignificant and any number of them may be entered between commands and parameters. However, in certain cases, one and only one blank must be entered - this blank is indicated by the "ß" character.

[...] Is used to specify an optional parameter. For example, [<VALUE>] would indicate that VALUE is an optional address parameter. You'll find that many parameters are optional, and in such cases logical default values will be supplied by BUG/65.

or Is used to delimit a list of choices. In such a list, one and only one choice may be used. For example, "+ or -" indicates that you may enter a plus sign or a minus sign, but not both.

filespec Is used to indicate a standard OS/A+ filespec. This consists of the device name followed by a colon and the filename. For example, "D:DATAFILE" is a valid filespec for a file named DATAFILE on disk drive one.

SECTION 3: Address Parameters

BUG/65 allows numerical addresses to be specified in a variety of ways. You can use hexadecimal or decimal notation, add and subtract terms, or add a relocation factor to any address. The following Backus-Naur definitions describe the various address types:

```
<ADDR>    :=  + or - <TERM>  [ + or - <ADDR> ]

<TERM>    :=  <NUMBER>  or  X<NUMBER>

<NUMBER>  :=  <DECNUM>  or  <HEXNUM>

<DECNUM>  :=  .<DECIMAL DIGITS>

<HEXNUM>  :=  <HEXADECIMAL DIGITS>
```

In the above, the only item not literally defined is the "X" item in the definition of a TERM. This is used to indicate that the following NUMBER is to be relocated by adding the value of the current relocation base to the value of NUMBER. The current relocation base is set by the "B" command.

All address parameters are interpreted as 16-bit positive numbers in the range of 0 to 65535. Overflow isn't detected or reported as an error.

Some examples will help (all of these are valid address expressions):

1FA1	a hexadecimal number.
.100	a decimal number (one hundred).
1000+.20	a hexadecimal number plus a decimal number. This evaluates to 1014 hex (4116 decimal).
1+2-3+4	a long expression. Evaluates to 4.
X1234	a relocated address. If the current relocation base has the value $1000, then this expression will evaluate to $2234.

3.1 Spaces as Parameter Delimiters

BUG/65 uses spaces as parameter delimiters. This makes for easier and quicker entry of commands. However, it does introduce some conventions regarding the use of spaces that you must be aware of:

* Spaces may not be embedded in a number. For example, "12 34" is interpreted as two parameters ($12 and $34) and not as the single parameter $1234.

* Spaces aren't allowed between the "X" relocation specifier and it's associated relocated address. For example, "X 1234" is interpreted as two parameters. The first will have the value of the current relocation base and the second is $1234.

* Any number of spaces may be used to separate two parameters. For example, "1234 5678" is a perfectly valid way of entering the two parameters $1234 and $5678.

SECTION 4: Loading and Running BUG/65

BUG/65 is shipped on your master diskette as a relocatable COMmand file, named "BUG65.COM". Therefore, BUG/65 functions just as does any OS/A+ extrinsic command: simply type "BUG65" when OS/A+ prompts with D1: (or Dn: if you have changed default drives...see the OS/A+ manual for more details) and BUG/65 will load into memory and relocate itself to just above the current value of LOMEM (contents of $2E7-$2E8).

4.1 Specifying BUG/65's Load Address

If you need BUG/65 to load at some location other than LOMEM (which is typically around $2000 with OS/A+ version 2 and around $2C00 with version 4), you may also enter a load address on the OS/A+ command line. The address must be in hex, must be at or below $9A00, and should be above LOMEM. Remember, BUG/65 occupies 8K bytes, which means it will occupy memory starting at the address you give and ending $2000 bytes higher.

EXAMPLE:

```
[D1:]BUG65 8000
```

This usage will load BUG/65 at $8000, set its restart point at $8200, and occupy memory from $8000 through $9FFF.

4.2 Creating a Non-Relocatable Version

In order to allow itself to be relocated virtually anywhere in memory, BUG/65 as shipped includes a relocation bit map and a relocation program. In addition, relocatable BUG/65 always loads in at locations $9800 through $BC00. If these addresses are "poison" to you (e.g., if you want to use BUG/65 with a cartridge plugged in), you may wish to produce a non-relocatable version designed to run within an address range you pick.

If so, USING A 48K SYSTEM, simply specify the loadpoint, as shown in the preceding section (e.g, via "BUG65 7000") and allow BUG/65 to load and relocate. Then exit to OS/A+ (via Quit) and use the OS/A+ intrinsic command SAVE to save a non-relocatable version. The address range to be SAVEd may be calculated as follows:

SAVE filename.COM loadpoint+$200 loadpoint+$2000

Thus, if you had specfied "BUG65 7000", you could save the non-relocatable version via

SAVE BUG7000.COM 7200 9000

thus also giving it a name which will later remind you where it will load at. To execute this non-relocatable version, simply type in its name (BUG7000 in the example shown).

SECTION 5: Command Entry

When you see BUG/65's input prompt (the ">" character) in the left-hand column of the screen, then you're in command entry mode. Any data typed at that point will be entered into the command line buffer - the command line isn't executed until you type RETURN. You can enter as many commands in one command line as will fit in the command line buffer (100 characters). As soon as you type the RETURN, you'll leave command entry mode and BUG/65 will begin executing the command(s) in the command line.

You can tell the difference between command entry mode and command execution mode. In command entry mode, the cursor is displayed. When a command is executing, the cursor is blanked.

If you try to enter more than 100 characters in the command line, BUG/65 will beep the bell and not allow any more characters to be input. At that point, you may either hit RETURN to execute what's in the command line so far, or edit some characters out of the command line with the BACKSPACE key.

5.1 Command Line Editing

When entering commands, you may edit mistakes with the BACKSPACE key. The BACKSPACE will move the cursor one column to the left and delete whatever character was in that column. Unfortunately, the normal system editing facilities aren't supported. This is because of the manner in which BUG/65 does keyboard input.

5.2 Normal and Immediate Type Commands

BUG/65 has two types of commands - normal and immediate. Normal commands are those that don't require interaction with the operator for their execution. Immediate commands do require operator interaction. Normally, you'll never be aware of the distinction between the two types - command entry "flows" without any consideration of the command type required. The only difference is that an immediate command must be the first command entered in a command line. Once an immediate command is entered, BUG/65 will begin interacting with the operator for further input. Since this interaction is required for completion of the command, it doesn't make sense to allow immediate commands to be "stacked" in the middle of a command line for execution between other commands. If you try to enter an immediate command in the middle of a command line, you'll get an "IMMEDIATE ERROR" error message and find yourself back in the command entry mode.

The immediate commands are the "A" command (ASCII memory change), the "S" command (hex memory change), the "X" command (change user registers), and the "Z" command (instant assembler).

5.3 Command Execution

For a normal type command, BUG/65 will begin command execution as soon as you type RETURN. For immediate type commands, BUG/65 will begin command execution as soon as you type the command character (provided that character is the first character in the command line).

5.4 Multiple Commands on a Line

Multiple commands may be entered on the same command line. Normally, successive commands in the command line don't require command separators between them other than at least one space character. The exceptions to this are commands for which an optional parameter is being defaulted. For example, the display memory command ("D") may have an optional parameter specified as the end of the area of memory to be displayed. If that ending parameter isn't specified, BUG/65 will default the end to the start plus eight bytes. If you wanted to enter two successive display commands in the command line without defaulting the end parameters, you could type

```
D 1000 1010 D 2000 2010
```

and no command separators would be required because BUG/65 knows that the "D" command only has two parameters and will interpret further characters in the command line as the beginning of a new command. However, if you wanted to default the ending address of the first display command, then you'd have to insert a command separator so that BUG/65 knows that the first display command is finished. If you didn't do this, then the second display command "D" would be interpreted as the second parameter of the first display command (the end address would be interpreted as $0D. The command separator is a comma, so in this case you would enter the commands as follows:

```
D 1000, D 2000 2010
```

SECTION 6: Command Termination

This section describes the many ways that a command will stop.

6.1 Normal Termination

Once a command line is given to BUG/65 for execution, BUG/65 will execute all of the commands in the line to conclusion before returning to command entry mode. It's possible to instruct BUG/65 to execute a command line "forever" (see Section 8.2), in which case BUG/65 will never come back to command entry mode until you manually intervene (with ESC or BREAK - see Section 6.4)

6.2 Error Termination

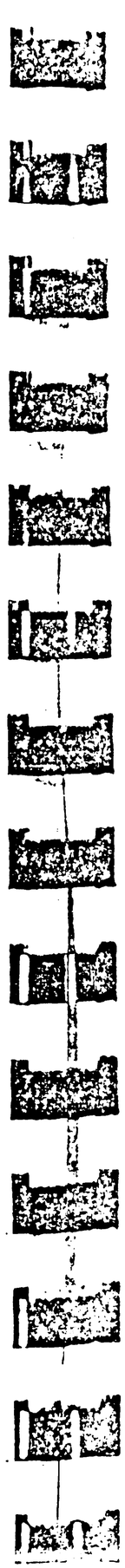
If an error occurs in command execution, BUG/65 will beep the bell and display a short error message in English indicating the cause of the error. Command execution will stop and you'll enter the command entry mode. Any commands in the command line after the command which caused the error won't be executed. (You should also be aware that BUG/65 will close any file that has been opened using IOCB number one when any error occurs.) (A complete list of error messages is in Section 14.)

6.3 Command Suspension

Once BUG/65 begins executing a command line, you may temporarily suspend command execution by hitting the space bar. This will put BUG/65 in a "hold" condition, at which point you have two alternatives: you can restart the command by hitting the space bar again, or you can abort the command with ESC or BREAK.

6.4 Comnand Abort

You can abort any command that is executing (except for the read and write disk commands) by hitting the ESC or BREAK keys. BUG/65 will stop executing the command and you'll enter command entry mode.



6.5 The RESET Key

BUG/65 traps the RESET key so that hitting RESET will bring you back to BUG/65. RESET will stop any command that is executing. You'll see the BUG/65 version and copyright prompt, and you'll be in command entry mode. RESET will reset all of BUG/65's internal stuff except for any user defined or modified parameters. For example, the user's registers, the current relocation base, etc., aren't cleared on a RESET - they'll retain whatever values they had before the RESET. (All of this depends, however, on the fact that the reset vectors haven't been modified by the user - either by using a BUG/65 command or by a user program. If you've modified the reset vectors, then the action of the RESET key is your responsibility.)

6.6 Manual Restart

Since BUG/65 is relocatable, the manual restart point (coldstart) depends upon where it has been relocated to. If you specified an address to load BUG/65 when you gave the OS/A+ command line (e.g., BUG65 4000), then the coldstart point is $200 greater than the address specified, and you may use 'RUN address' from OS/A+ if desired (e.g, RUN 4200 if the original command was BUG65 4000). In any case, you may inspect location $000C (via the BUG/65 command 'D C') to determine the coldstart point. The 6502 word address in locations $0C and $0D (LSB, MSB order) points to BUG/65's restart point. The result of a manual restart is the same as if the default RESET key processing occurred (see section 6.5).

SECTION 7: Command Descriptions

Throughout the descriptions of the commands, comments are sometimes presented in the command line examples. These are denoted by the characters "*/". Anything appearing on a line after these characters is a comment and is NOT part of the command line being exemplified.

The commands are presented in alphabetical order.

7.1 A - Change Memory, ASCII mode

A <ADDR>ß

The A command allows you to replace the contents of memory bytes beginning at location <ADDR> with ASCII characters. As soon as you type the required space character after the address, BUG/65 will prompt you with the current contents of the memory location at <ADDR>. Those contents will be displayed as an ASCII character. At that point, you have the following options:

1. Typing a SPACE will cause the current memory location to be skipped and the contents of the next memory location to be displayed.

2. Typing an UNDERLINE will cause the current address to be decremented by one. The new address is then displayed on the next line of the screen followed by the contents of the new memory location.

3. Typing a RETURN will cause the address of the current memory location to be displayed on the next line of the screen followed by the contents of the current location.

4. Typing ESC will get you out of the command and back into command entry mode.

5. Typing any character other than "@" will cause the ATASCII value of that character to be entered into memory at the current address. The address is then incremented by one and the contents of the new memory location are displayed.

6. Typing the character "@" causes the next character typed to be entered into the current memory location as its pure ATASCII value without any of its control character significance. For example, typing "@ ESC" will insert the ATASCII value for ESC into memory. The address is then incremented by one and operation continues as in 5. above.

After you exercise any option except option 4., BUG/65 will again prompt you with the contents of the current location and you may then choose from any option again.

7.2 B - Set Relocation Base

B <ADDR>

The B command will set the value of the relocation base to ADDR. The relocation base is intended for use with relocating assemblers. In a relocatable environment, listings typically are addressed from location zero. When a module to be debugged is subsequently loaded into memory, it will have a relocation offset added to the addresses in the listing. The B command allows you to set the relocation base to the load address of the module you're working on and then to reference addresses within the module by simply prefixing each address expression with the relocator symbol "X".

For example, suppose that a relocatable module is loaded at location $5380 in memory. Suppose further that we want to display the contents of a memory location which is $230 from the beginning of the module. The following commands would do the job:

```
B 5380, D X230
```

The world isn't overrun with relocating assemblers for the ATARI. However, until it is, the B command has other useful applications. These take advantage of the fact that the relocation base value is a variable which can be modified during command execution. For example, suppose you know that the string of characters "ABCD" is stored somewhere on a diskette and you want to find the sector that contains it. The following commands will do the trick:

```
B 1

D X, R$ X 4000 1, L 4000 407F 41 42 43 44, B X+1/
```


This uses some commands not introduced yet, but this is what happens: First, X is set to 1 with one command line. Then a second command line will display memory at the location X (so you'll know where you're at as you step through), read sector number X into memory locations $4000-$407F, locate the string "ABCD" in that sector buffer, then bump X by one for the next sector. The slash at the end of the command line means that the command line will execute forever. What will happen is that BUG/65 will continuously read diskette sectors. For every sector read, you'll see at least a memory display of eight bytes beginning at address X (which is the sector number). If the Locate instruction finds the string "ABCD" in the sector buffer, it will display the location of the string. At that point, just hit ESC to stop the command, and display the value of X ("D X RETURN"). The sector containing the string will either be the value of X or one before it, depending on how fast your ESC was.

7.3 C - Compare Memory Blocks

C <STARTBLOCK1> <ENDBLOCK1> <STARTBLOCK2>

Compare is used to compare the contents of two blocks of memory. The block of memory beginning at STARTBLOCK1 and ending with ENDBLOCK1 is compared to the same size block beginning at STARTBLOCK2. If both blocks are the same, then there will be no output. If any bytes in the blocks differ, then BUG/65 will display a line of data in the following format for every byte that is different:

```
AAAA = BB CCCC = DD
```

where AAAA = the hex address of the differing location in the first block, BB = the hex contents of location AAAA, CCCC = the hex address of the differing location in the second block, and DD = the hex contents of location CCCC.

7.4 D - Display Memory

D <START> [<END>]

The D command displays the contents of the memory block beginning at START and ending at END. If END isn't specified, then the default value of START+7 is used. The memory block is displayed in the following format:

AAAA = BB BB BB BB BB BB BB BB CCCCCCCC

where AAAA = the hex address of the first byte in this line, BB = the hex contents of successive memory locations beginning at location AAAA, and C = the ASCII character interpretation of the positionally corresponding BB value of the byte.

7.5 E - Execute a Command File

E #filespec

The E command is used to execute a command line from a command file. The file specified by filespec must consist of a line of BUG/65 commands and parameters and must be ended with an ATASCII EOL character ($9B). BUG/65 will only execute one command line from a command file and then it will stop reading the file. Command files can be chained however, so that the last command in one file can execute another command file. An E command should be the last command in a command line because any commands after the E in the line won't be executed.

7.6 F - Fill a Memory Block with a Value

F <START> <END> [<VALUE>]

The F command will fill the block of memory beginning with START and ending with END with VALUE. If VALUE isn't specified, then zero will be used. Note that VALUE is a byte value - the least significant byte of the 16-bit VALUE will be used for the fill.

7.7 G - Goto a User Program

G [<START>] [@<BRKPOINT> [RN=<VALUE>] [I=<COUNT>]]

The G command will execute a user program beginning at START. If START isn't specified, then execution begins at the current value of the user's PC register. BRKPOINT is an optional breakpoint. If the user's program trys to execute the instruction at BRKPOINT, the program will break back to BUG/65 and BUG/65 will display the contents of the user's registers at that point. Examples:

```
G  1000    /* go at location $1000, no breakpoint

G  @4300   /* go from wherever our PC was and break
           /* at location $4300
```

A breakpoint may be conditionally qualified by a required value in a specified register. "RN=<VALUE>" will tell BUG/65 to break at that point only if the value of user register "N" equals VALUE. If that condition isn't met, then the user's program is allowed to continue executing at the location of the breakpoint. (The instruction that was at the breakpoint location WILL be executed.) The mnemonic names of the registers that may be specified for "N" are: A, X, Y, S, and P, which stand for the user's A, X, Y, Stack, and Status (flags) registers respectively. (Note that only the least significant byte of VALUE is used for this qualification.)

Example:

```
G  1000  @1422  RX=33  /* go from location $1000 and
                      /* break at location $1422
                      /* only if register X equals
                      /* $33
```

A breakpoint may also be qualified with an iteration counter. "I=<COUNT>" tells BUG/65 to allow the execution of the instruction at the breakpoint COUNT times before breaking.

Example:

```
G   1000  @2300  I=2  /* go from location $1000 and
                     /* break the second time we hit
                     /* the instruction at $2300
```

The register and iteration qualifications may be used together. In this case, the register condition must be met before the iteration counter is decremented. As in the following example:

```
G  1000  @1234  RA=50  I=3  /* go from location $1000
                            /* and break the third time
                            /* the instruction at loc-
                            /* ation $1234 is executed
                            /* with register A equal
                            /* to $50
```

All of this flexibility isn't without its price, however. Because BUG/65 has to do quite a bit of evaluation at every breakpoint before deciding if the break condition has been met, don't expect to be able to conditionally pass through breakpoint instructions at real-time speed. As long as you never execute the instruction at the breakpoint, you're OK, but as soon as BUG/65 gets the break, expect several hundred instructions to be executed before your program is given back control after the break isn't met.

Also, BUG/65 was NOT designed to allow breakpoints in PROM resident code. If you attempt to set such a break point, or if you try to set a breakpoint at a non-existent memory location, you'll get a "BREAKPOINT ERROR".

One other thing. BUG/65 will automatically remove breakpoints from your program after a break occurs. Breakpoints aren't left set after the break is performed.

7.8 H - Hexadecimal Arithmetic

H <NUMBER1> <NUMBER2>

The H command will calculate the sum NUMBER1 + NUMBER2 and the difference NUMBER1 - NUMBER2 and display the results on the next line of the screen as two hex words. The sum is the first word displayed, the difference is the second.

7.9 I - Display Disk Directory

I

The I command will display the directory of the diskette in drive one. The display can be suspended or halted with the SPACE or ESCAPE keys respectively.

7.10 J - Create a Command File

J #filespec, string

The J command allows you to create command files for execution by the E command. The string in the command is any string of valid BUG/65 commands. The string will be written to the file specified by filespec in the format expected by the E command. Please note the comma after the filespec - it's required, else BUG/65 won't know where your filespec stops and your command string starts. Also note that the J command doesn't allow multiple commands in the command line to be executed after the J command - everything in the line after the filespec and up to the RETURN is written to the file instead of being executed.

7.11 K - Convert Hex to Decimal

K <NUMBER>

The K command will convert NUMBER to a decimal number and display the result on the next line of the screen. NUMBER can be any valid address expression.

To convert decimal to hex, just display memory at the decimal location of the number you want to convert. The hex equivalent of the decimal location appears in the display output as the hex word on the beginning of the line. For example, to convert 1000 decimal to hex, just execute the command "D .1000". You'll see the hex conversion of 1000 as the first hex word on the next line.

7.12 L - Locate a Hex String

L <START> <END> <BYTE1> <BYTE2> ... <BYTEn>

The L command will search the block of memory beginning at START and ending at END for a hex string. The hex string is defined by BYTE1...BYTEn, which are interpreted as the hex bytes of the pattern string. (Only the least significant bytes of the address values are used for each byte in the string.) Wildcard bytes which will match any byte in memory may be specified by the character "*" in the string. BUG/65 will output the addresses of every occurrence of the string found in the block. For examples:

```
L 1000 10FF  41  42  43 /* will locate any occur-
                        /* rences of the string "ABC"
                        /* in the memory block
                        /* $1000 to $10FF

L 1000 2000  10  *  20  /* will locate any occur-
                        /* rences of a three-character
                        /* string which begins with
                        /* $10 and ends with $20 in
                        /* the memory block $1000
                        /* to $2000
```

7.13 M - Move a Memory Block

M <START> <END> <TO>

The M command will move the block of memory beginning at START and ending at END to TO. BUG/65 will take care to handle overlapping moves correctly, either for moves up or down.

7.14 P - Select Output Devices

P [S] [P]

The P command is used to select output to either the screen ("S") or the printer ("P") or to both ("SP"). For example:

```
P  S       /* turns screen output on, printer output off
P  P       /* turns printer output on, screen output off
P  S  P    /* turns both screen and printer output on
P          /* turns both outputs off - commands will
           /* still be accepted and executed, you just
           /* won't see their entry or output anywhere.
```

In addition to allowing you to list BUG/65 results to the printer, this command was designed to allow you to debug the generation of intricate screen displays without having the outputs of BUG/65 commands scroll your display off the screen. It is a little crude, and might have a few problems depending on what your program has done to OS, but is handy to have in emergencies. (The LFFLAG and NULFLG bytes in the Configuration Table can help you here - see section 11.)

7.15 Quit to OS/A+ command

Q

The Q command will coldstart DOS. The results are essentially the same as when you power-up the machine.

7.16 Read Commands

7.16.1 R - Read a File

R [<OFFSET>] #filespec

The R command is used to load binary files. If OFFSET is specified, then OFFSET is added to the load address(es) specified in the file, and the data will be loaded at the loading point(s) plus OFFSET. This allows you to load a file into a different memory location than where it is origined at. After the file is loaded, the load starting point specified in the file is placed into the user's PC register.

BUG/65 supports concatenated binary file sections as described in the DOS 2.0S manual. If such a file is loaded using the OFFSET option, however, ALL file sections will be loaded starting at the load addresses specified in the file plus OFFSET. In addition, the user's PC register will contain the value of the load point of the last file section loaded (not plus OFFSET).

7.16.2 R% - Read Sector(s)

R% [<SECNO> [<BUFFER> [<NOSECS>]]]

The R% command allows you to read a sector or a group of sectors from a diskette in disk drive number one. SECNO specifies the sector number to be read and defaults to one. BUFFER specifies the buffer the sector is to be read into and defaults to BUG/65's loadpoint plus $2000. NOSECS specifies the number of sectors to read and defaults to one. If more than one sector is specified, then consecutive sectors are read sequentially into memory beginning at BUFFER.

7.17 S - Change Memory, Numeric mode

S <ADDR>þ

The S command allows you to replace the contents of memory bytes beginning at location ADDR with numerical values. As soon as you type the required space character after the address, BUG/65 will prompt you with the current contents of the memory location at ADDR. Those contents will be displayed as a hexadecimal byte value. At that point, you have the following options:

1. Typing SPACE will cause the current memory location to be skipped and the contents of the next memory location to be displayed.

2. Typing an UNDERLINE will cause the current address to be decremented by one. The new address is then displayed on the next line of the screen followed by the contents of the new memory location.

3. Typing a RETURN will cause the address of the current memory location to be displayed on the next line of the screen followed by the contents of the current location.

4. Typing ESC will get you out of the command and put you back into command entry mode.

5. Typing an address value (any valid address expression) will cause that value to be entered into memory at the current address. The address is then incremented by one and the contents of the new memory location are displayed. (Only the least significant byte of the address value will be entered into memory.)

After you exercise any option except option 4., BUG/65 will again prompt you with the contents of the current memory address and you may select any of these options again.

7.18 T - Trace a User Program

T [S] [<COUNT>]

The T command will single-step through user program instructions beginning with the instruction at the current user PC register. The number of instructions to be executed are specified by COUNT, which defaults to one. If "S" is specified, then all of the instructions in a subroutine are counted as one instruction for tracing purposes - the trace is turned off until return from the subroutine ("S" stands for "skip the subroutine"). After every instruction traced, BUG/65 will display the contents of the user's registers. Some examples:

```
T           /* will execute one instruction and then
            /* display the register contents

T 5         /* will execute five instructions, displaying
            /* registers after each instruction

TS 10       /* will execute 16 instructions. If any of
            /* the instructions are JSR's, then the
            /* trace will be turned off after the JSR
            /* until the subroutine executes an RTS
```

The trace command can't be use to trace instruction execution through PROM resident code. Any attempt to do so, or to trace through non-existent memory, will result in a "BREAKPOINT ERROR".

7.19 U - Call a User Subroutine

U <ADDR> [<PARAM>]

The U command is used to call a user subroutine at ADDR. The user routine is passed the optional parameter PARAM in the X register (low byte) and Y register (high byte). The user routine should return to BUG/65 via an RTS instruction. If PARAM isn't specified, then zero is used.

7.20 V - Display User's Registers

V

The V command will display the contents of the user's registers in the following format:

```
 A  X  Y SP NV_BDIZC   PC  INSTR
HH HH HH HH BBBBBBBB HHHH  LDA  1000,X
```

This is interpreted as follows:

```
A  = the hex value of the A reg
X  = the hex value of the X reg
Y  = the hex value of the Y reg
SP = the hex value of the stackpointer
N  = the binary value of the negative flag
V  = the binary value of the overflow flag
_  = the binary value of an unused bit in the
       status reg
B  = the binary value of the break flag
D  = the binary value of the decimal flag
I  = the binary value of the interrupt enable bit
Z  = the binary value of the zero flag
C  = the binary value of the carry flag
PC = the hex value of the PC reg (This is a
       pseudo register maintained by BUG/65.
       It contains the location of the next
       user program instruction to be executed.)
INSTR = the instruction at the current PC
```

7.21 Write Commands

7.21.1 W - Write a File

W [:A] <START> <END> #filespec

The W command is used to write a binary file. Memory from START to END is written to the file specified by filespec in the standard OS/A+ binary file format. If the ":A" option isn't specified, then the data written will replace the current contents of the file if the file already exists. If the ":A" option is specified, then the data is appended to any data already in the file. A load header consisting of a start and end address as described in the OS/A+ manual will precede the appended data.

7.21.2 W% - Write Sector(s)

W% [<SECNO> [<BUFFER> [<NOSECS>]]]

The W% command is used to write a sector or a group of sectors to a diskette. SECNO specifies the sector number to be written and defaults to one. BUFFER specifies the memory location of the sector data to be written and defaults to the BUG/65 loadpoint plus $2000. NOSECS specifies the number of sectors to be written and defaults to one. If more than one sector is specified, then consecutive sectors are written sequentially from memory beginning at BUFFER.

7.22 X - Change User's Registers

X REGNAME

The X command allows you to change the contents of user registers. REGNAME is a one-character register name mnemonic. The allowed register names and their meanings are:

A = A register
X = X register
Y = Y register
S = stackpointer register
P = program counter pseudo-register
F = status register (flags)

After you type in the name of the register to be changed, BUG/65 will prompt you with that name character followed by an equals sign. At that point you have the following options:

1. Enter the new value for the register. The new value may be any valid address expression. After the new value, typing RETURN will end the command. Or you can type SPACE which will prompt you with another register name for possible change. The next register name is determined by the order of the above list. For example, if you change register Y then hit a space after the new value, BUG/65 will prompt you for possible change of register S. This prompt list continues through register F and then wraps back to register A again.

2. Enter RETURN or ESC to end the command. BUG/65 will display the new contents of the registers and then put you back into command mode.

7.23 Y - Disassemble Memory Block

Y <START> <END>

The Y command will disassemble instructions in memory beginning at START and ending at END. The following conventions are used in the disassembly:

1. Standard MOS Technology mnemonics are used for opcodes.

2. Illegal opcodes are displayed as "***".

3. All numeric operands are displayed as hexadecimal numbers.

4. Zero page operands will display as two hex digits, all other non-immediate operands will display as four hex digits.

5. No operand is displayed for accumulator mode operands.

7.24 Z - Instant Assembler

Z <ADDR>Ø

The Z command allows you to assemble instructions to be stored in memory at ADDR. Immediately after typing the SPACE character (or RETURN, which is allowed as well), BUG/65 will prompt you with the current program counter value of the instant assembler (which initially will be ADDR). At that point you may type in a valid assembly language instruction. The format for an instruction line is:

[<LABEL>] <OPCODE> [<OPERAND>]

LABEL may be any label in the form "Ln", where "n" may be any digit from zero to nine. OPCODE may be any valid MOS Technology instruction mnemonic or one of two pseudo-ops (described below). OPERAND, if allowed by the addressing mode of the instruction, may be any valid address expression. At least one space must separate a label from an opcode or an opcode from an operand.

After typing your instruction, type RETURN and the instruction will be entered into memory at the current PC if it doesn't contain any errors. If there are any errors, then BUG/65 will display an error message and will reprompt you with the current (unchanged) PC. If there are no errors, then BUG/65 will display the object code created by the instruction to the right of the instruction on the screen and will prompt you with the PC of the next instruction on the next screen line. You may exit the instant assembler by typing ESC at any time, or by typing RETURN by itself in response to the PC address prompt.

The instant assembler provides you with two pseudo-ops. "/" followed by an address will change the PC to that address. It acts like an ORG ("*=") pseudo-op. For example, "/4000" will set the PC of the next instruction location to $4000. "+" followed by an address will insert the value of that address (least significant byte) at the current PC and bump the PC by one. It acts like a DB (.BYTE) pseudo-op. For example, "+34" will insert the hex byte 34 at the current PC.

The instant assembler provides a simple labeling capability. You may prefix an instruction with a two character label of the form "Ln", where "n" may be any digit from 0-9. You may then use that label as an operand in an instruction, with the following three restrictions:

1.) Immediate type operands (#HH) can't be labels.

2.) Indirect type operands can't be labels.

3.) A label can't be combined with any of the standard address operators (+, -, X, etc.)

Label references may be forward or backward. BUG/65 will store unresolved references and resolve them when the label is later defined. You may reference undefined labels twenty times before BUG/65 runs out of room to store the unresolved locations - you'll then get an error message and the assembly will be aborted. The same label may be reused more than once. In such cases, BUG/65 will use the last defined address of the label when it is referenced.

If any labels have been referenced but not defined when you exit the instant assembler, BUG/65 will prompt you with a message and the label name followed by an equals sign. At that point you may either define the label by entering any valid address expression followed by a RETURN, or you may chose not to define it and simply hit RETURN. If you don't define the label, then the value of the label is defaulted according to the following two rules.

1.) If an instruction using the undefined label is a relative branch, then the value of the label for that instruction defaults to the location of the instruction plus two.

2.) For all other instructions, the value of the label defaults to the location of the instruction plus three.

These rules guarantee that all branching instructions using undefined labels are effectively turned into NOP'S. This offers some measure of protection against a program going into never-never land. (If you reference a label that isn't yet defined, the object code displayed to the right of the instruction on the screen will show addresses generated according to these rules. Don't worry, when the label is subsequently defined, BUG/65 goes back and fixes up all these references.)

SECTION 8: Special Command Modifiers

8.1 Repeat Last Command Line

[RETURN]

The last command line entered and executed may be repeated without typing the whole thing in again - just hit RETURN. BUG/65 remembers the last line entered for just this purpose.

8.2 Repeat Command Line Forever

/

Appending a slash to the end of a command line will cause BUG/65 to repeat the execution of that command line forever. The only way to stop such a repeat is to suspend or abort the command.

8.3 Display Last Command Line

=

If you want to see what your last command line was, possibly because you might want to repeat it, just type the "=" character as the first character of the new command line. BUG/65 will display the last line entered for you.

SECTION 9: BUG/65 Memory Protection

BUG/65 won't allow you to modify any portion of it's code or variable storage areas with a BUG/65 command. Any attempt to do so will result in a "PROTECTION ERROR". For example, if we assume that the BUG/65 was loaded via the command "BUG65 2000", the following command will cause an error because it attempts to move a memory block into BUG/65's area:

```
M 4000 40FF 2000
```

BUG/65 protects all memory from loadpoint to loadpoint+$1FFF in this manner, where loadpoint is that specified in the invoking OS/A+ command line (or LOMEM, if no loadpoint is specified). (The memory protection feature can be turned off by changing a byte in the Configuration Table.)

SECTION 10: BUG/65 Memory Usage

BUG/65 uses memory from $80 to $XX and loadpoint to loadpoint+$01FF for variable storage. You can determine the value of XX by looking at the LSTPG0 byte in the Configuration Table. It uses memory from loadpoint+$200 to loadpoint+$1FFF for code storage.

10.1 Page Zero Sharing

BUG/65 will share the page zero memory that it needs with a user program. It does this by keeping two copies of these page zero locations. When BUG/65 is running, the BUG/65 page zero locations contain BUG/65's stuff. When a Go is done to a user program, BUG/65 will save it's own page zero data and replace it with the user's data. If a user program breaks back to BUG/65, the reverse operation is performed.

In addition, BUG/65 will translate any command reference to these shared page zero locations so that the user may modify or inspect his own page zero data. It does this by translating any command reference to the user's page zero data to the location where the user's copy of the data is actually being stored. This is all transparent to the user. For example, you can fill memory from $80 to $FF with zeros without crashing BUG/65. If you then display $80 to $FF, you will see zeros. They aren't really in locations $80 to $FF of course, but they will be when you run your program. (This is the reason it may seem to take an extraordinarily long time to perform certain commands (Fills, for example). The reason is that every memory reference has to go through this translation process - both to translate zero page references if necessary and to check to make sure that BUG/65 isn't being overwritten.)

SECTION 11: Customization with the Configuration Table

There is a Configuration Table located near the beginning of the code segment of BUG/65. By changing this data, you can customize some BUG/65 stuff. In the table which follows, "+$xxx" means that the configuration value is located $xxx bytes above the loadpoint address, where loadpoint is the address specified in the invoking OS/A+ command line (or LOMEM, if loadpoint is not specified). Example: if the invoking command was "BUG65 6000", then DISPV will be located at $6209.

NAME	LOCATION	FUNCTION/COMMENTS
DISPV	+$209	A JMP instruction to BUG/65's display a character routine. All chars displayed on the screen go through here. The char to be displayed is passed in reg A.
PRINTV	+$20C	A JMP instruction to BUG/65's print a character routine. All chars sent to the printer go through here. The char to be printed is passed in reg A.
GETKYV	+$20F	A JMP instruction to BUG/65's get a keyboard character routine. All keyboard reads go through here. The key read is returned in reg A.
TSTKYV	+$212	A JMP instruction to BUG/65's test for a key waiting routine. All tests for key waiting go through here. If no key is waiting, the equal flag is returned set. (The key is NOT returned by this routine - GETKYV will be called to read the key if there's one waiting.)
BEEPV	+$215	A JMP instruction to BUG/65's bell routine. All beeps are generated through here. To eliminate the beeps, just patch this out with an RTS.
CHRCLR	+$218	Character background color byte value.
CHRLUM	+$219	Character luminance byte value.
BRDCLR	+$21A	Border color byte value.
EOLBYT	+$21B	This is the byte sent to the printer at the end of a line. Normally set to ØDH or 9BH.

LFFLAG	+$21C	If nonzero, then a linefeed character is sent to the printer after every EOLBYT.
NULFLG	+$21D	If nonzero, then 40 nulls will be sent to the printer after every line. Used to flush the printer buffer maintained by the ATARI OS so that all lines will print immediately.
PROTFG	+$21E	If nonzero, then BUG/65 will not allow itself to be overwritten with a BUG/65 command. If zero, then BUG/65 will allow itself to be modified.
MCBEND	+$21F	High byte of end address of BUG/65's code. Normally set to high byte address of loadpoint+$2000 (e.g, $50 if the invoking OS/A+ command were BUG65 3000). You would change this if you added any user command handlers after BUG/65. The handlers would then be included in BUG/65's memory protection features.

To change anything in the Configuration Table, you must first disable memory protection by writing a small program to stuff a zero into PROTFG. For example, assuming that the loadpoint is $2000 (command line was BUG65 2000), then using the instant assembler, you could enter "LDA #0, STA 221E, RTS" at location $5000, and then run the program with the "U" command by entering "U5000 <RETURN>". This will disable memory protection. Then make your changes, reenable memory protection if you want by storing $FF into PROTFG, then dump the modified BUG/65 to diskette.

Be careful when changing any of the JMP instruction vectors. Since BUG/65 is constantly calling these locations, the instant you change them control will be passed to the new routine. Your replacement routines had better be in place and ready to run or it's ga-ga time. Actually, you will probably have to change all three bytes of a vector at once with a small user program.

Also, be careful about calling the vectors DISPV, PRINTV, GETKYV, TSTKYV, and BEEPV. Since they use BUG/65's page zero data to operate, they can't be called from a running user program without first calling the MCBGPØ routine defined in the User Program Interface section.

SECTION 12: User Command Interface

It's possible to add commands to BUG/65. The hooks to do so have been provided in a group of vectors located at loadpoint+$0220 called the User Command Interface Vectors. These vectors provide most of the interfaces to BUG/65 that you'll need to add commands.

The commands you add may be activated by any non-BUG/65 command char. For example, you could add the numeric commands "1" through "9". When BUG/65 recognizes a non-alphabetic command character, it will call the vector USERCMD. In it's initial state, USRCMD is just a 3-byte subroutine that returns the equal flag reset. BUG/65 assumes that the equal flag being reset means that a user command handler considers the command illegal. In this case, BUG/65 will report a "CMD ERROR". If USRCMD returns the equal flag set, then BUG/65 assumes that a user command handler processed the command. In this case, BUG/65 won't generate a command error, and will proceed to process the rest of the command line.

So, to add your own command handler, just patch a JMP to your handler at USRCMD. BUG/65 will pass you the command character that it considered illegal in reg A. On return, you must indicate the status of the command - equal set means you handled it, equal reset means you didn't like it either.

There are a number of other vectors in the User Interface group which you may use to process the command. Here's the complete list (and, as in the previous section, the string "+$xxx" indicates a displacement from the loadpoint):

NAME	LOCATION	FUNCTION/COMMENTS
USRCMD	+$220	Subroutine called by BUG/65 on every non alpha comand char. Returns equal set if command handled by user, else equal reset.
GETCHR	+$223	User handler can call this to get the next char from the command line in reg A.
PUTCHR	+$226	User handler can call this to return the last char taken from the command line. The char itself doesn't have to be passed. This is used to put chars back that you've taken but don't want - like an EOL.
GET1HX	+$229	User handler can call this to collect a hex address from the command line. The address is returned in a word at $FE,$FF. If next command line chars are not a valid address, zero is returned.
GET2HX	+$22C	User handler can call this to collect two hex addresses from the command line. The first address is returned in a word at $FC,$FD, the second at $FE,$FF. Zero is returned for any invalid address.
GET3HX	+$22F	User handler can call this to collect three hex addresses from the command line. The first address is returned in a word at $FA,$FB, the second at $FC,$FD, and the third at $FE,$FF. Zero is returned for any invalid address.

ADRCHK	+$232	User handler can call this to perform the usual BUG/65 address checking and translation. The checking refers to not allowing BUG/65 to be overwritten. The translation refers to correcting user page zero addresses. The user handler passes the address to check in reg X (LO) and reg Y (HI). If the address points into BUG/65, a "PROT ERROR" will occur, and the user handler will not be returned to. If the address references a user page zero value that is being stored somewhere else by BUG/65, then the address of where the actual user page zero byte is located will be returned in reg X (LO) and reg Y (HI).
ERRPAR	+$235	The user handler can JMP to here to report a parameter error. There is no return back to the user handler. BUG/65 will abort command line processing.
DHXBYT	+$238	The user handler can call this to display a hex byte. The byte is passed in reg A.
DHXWRD	+$23B	The user handler can call this to display a hex word. The hex word is passed in reg X (LO) and reg Y (HI).
CTBPTR	+$23E	This is a pointer to BUG/65's jump table for the alphabetic comands. Every letter has a word entry in this table. The entry is the address of the handler for that command minus one. The first word in the table is the address minus one for the "A" command, the last is the same for the "Z" command. If you want, you can change this table to point to your own comand routines, thereby changing the BUG/65 command set.
LSTPG0	+$240	This is the address (byte value) of the last page zero location used by BUG/65. You can use this to locate free page zero memory for your own use. (See the example user command listing.).

**** SPECIAL NOTE ****

All of the above routines assume that BUG/65 data is in page zero. THEY WILL NOT WORK if called from a running user program for that reason, unless the user program manages page zero with the following two routines:

MCBGP0	+$241	Assumes BUG/65 data is in page zero. Saves BUG/65 page zero and replaces with user page zero. Use this routine from a running user program before calling any of the above routines.
USERP0	+$244	Assumes user data is in page zero. Saves user page zero and restores BUG/65 page zero. Use this routine from a running user program after calling any of the above routines to restore the running program's page zero data.

12.1 User Command Handler Example

Here is an assembly listing of an example user comand. This command will be command "1". It will calculate and display an exclusive-or checksum byte on a range of memory. The syntax of the command is:

```
1 <START> <END>
```

NOTE: It is highly recommended that user commands only be patched into a non-relocatable version of BUG/65. See Section 4.2 for instructions on making a non-relocatable version with a user specified loadpoint.

```
;*****************************************************
;
;  EQUATES INTO BUG/65:

loadpoint =  ????              to be determined by user!!
lp = loadpoint                 just an abbreviation

MCBEND  =    lp+$21F           BUG/65 END CODE MSB
DISPV   =    lp+$209           DISPLAY CHAR
USRCMD  =    lp+$220           USER COMMAND VECTOR
GET2HX  =    lp+$22C           GET 2 HEX PARAMS
HEX1    =    $FC               HEX PARAM 1 RESULT
HEX2    =    $FE               HEX PARAM 2 RESULT
ERRPAR  =    lp+$235           REPORT PARAM ERROR
DHXBYT  =    lp+$238           DISPLAY HEX BYTE
LSTPG0  =    lp+$240           LAST BUG/65 P0 BYTE USED
;
EOL     =    $9B               END OF LINE CHAR
```

```
;*************************************************
          *=    USRCMD          PATCH US INTO BUG/65
          JMP   USERC1
;
          *=    1p+$2000        RIGHT AFTER BUG/65 CODE
USERC1    CMP   #'1             COMMAND "1" ?
          BEQ   CMDOK           YES
          RTS                   ELSE RTN EQUAL RESET - ERR
;
CMDOK     JSR   GET2HX          GET START, END
          LDA   HEX1            MAKE SURE BOTH SPECIFIED
          ORA   HEX1+1
          BEQ   PARMER          OR ELSE ERROR
          LDA   HEX2
          ORA   HEX2+1
          BNE   PARMOK
;
PARMER    JMP   ERRPAR          REPORT PARAM ERROR
;
PARMOK    LDX   LSTPG0          LAST BUG/65 P0 BYTE
                                  (WE'LL USE THE NEXT
                                  FOR OUR ACCUMULATOR)
;
;
          LDA   #0              CLEAR ACCUMULATOR
          STA   1,X
          TAY                    INIT Y PTR INDEX
;
LOOP      LDA   HEX2+1           PAST END ADDRESS ?
          CMP   HEX1+1
          BCC   DONE             YES
          BNE   NXTEOR           NO
          LDA   HEX2
          CMP   HEX1
          BCC   DONE             YES
;
NXTEOR    LDA   (HEX1),Y         CALC EOR CHKSUM
          EOR   1,X              EOR WITH ACCUM
          STA   1,X              AND SAVE IN ACCUM
          INC   HEX1             BUMP PTR
          BNE   LOOP
          INC   HEX1+1
          JMP   LOOP
;
DONE      LDA   #EOL             TO NEXT SCREEN LINE
          JSR   DISPV
          LDX   LSTPG0           RESTORE ACCUM ADDRESS
          LDA   1,X              DISPLAY HEX RESULT
          JSR   DHXBYT
          LDA   #0               RTN OK (EQUAL SET)
          RTS
;
          *=    MCBEND           CHANGE BUG/65 CODE
          .BYTE >[*+$FF]         END BYTE TO INCLUDE
          .END                  THAT'S ALL FOLKS
```