Debug+

by Bryan Schappel

ANALOG

Analog Computing

February 1986

Debug+ is a screen-oriented machine language debugging utility. It contains a program tracer that can step through almost any machine language program in three different ways. **Debug+** also has a complete scrolling disassembler and memory "dumper." It allows user program execution and can perform binary SAVEs and LOADs, plus many, many more functions. Sound too good to be true for a magazine program? Well, it's not. Read on!

Typing it in

Before you start typing anything, examine the listings accompanying this article. **Listing 1** is the main data and data checking routine, written in Atari BASIC. This program will create a file called DEBUG.COM. Follow the instructions below to create the DEBUG. COM file.

1. Type Listing 1 into your Atari and verify it with **Unicheck** (see page 11).

2. Type *RUN* and press RETURN. The program will begin checking the data lines, printing the line numbers as it goes. You'll be alerted if it finds any problems. Fix any incorrect lines and re-RUN the program as necessary, until all errors are eliminated.

3. When all data lines are correct, you will be prompted to *INSERT DISK WITH DOS, PRESS RETURN.* Place a disk in drive 1with DOS and press RETURN. The message *WRITING FILE* will appear, and the computer will create the DEBUG.COM file, printing the line numbers as it goes. Make sure you save the BASIC program under a different filename before continuing.

To LOAD **Debug+** from Atari DOS 2.0S, go to DOS and type:

```
DEBUG.COM
```

Debug+ will LOAD and RUN automatically.

For those interested in assembly language, the OSS MAC/65 source code for **Debug+** is available on the **ANALOG Computing** TCS and is included on the disk version of this issue.

Remember, you must have the BASIC cartridge removed for **Debug**+ to LOAD. For XL owners, this means you must boot up while holding down the OPTION key. The program resides in cartridge slot A memory locations ($A100 - $C0FF) and uses addresses $BE00 - $C0FF as screen memory. The program supports eighteen commands, which are listed below:

Key	**Function**
*	Address Set
D	Display Toggle
Q	Quit DEBUG+, go to DOS
G	Go at address
T	Trace Program
P	Print Disassembly
E	Erase Memory
C	Change 1 byte of RAM
N	Change Register Value
R	Display Registers
B	Set/Reset Break Point
S	Save a binary file
L	Load a binary file
F	Find a string in memory
H	High Speed Display
"	Dec/Hex, Hex/Dec convert
-	Scroll up in memory
=	Scroll down in memory

Before I explain all the functions, I'd like to tell you about the prompts that **Debug+** uses. There are three basic prompts: **, ?* and *(opt1, opt2, opt3)?*

When the * appears in the input window, **Debug+** expects a number. This number may be entered in either hexadecimal or decimal. To enter a decimal number, the digits in the number must be preceded by a decimal point (a period). If the number is not preceded by a period, **Debug+** will interpret it as hexadecimal. If there are any "illegal" digits in the number, **Debug+** will respond with a short tone, and the command will be aborted.

When the *?* appears, **Debug+** expects a string of characters, a filename, for example, with a maximum of twelve characters. When you reach the maximum input length, all extra characters (except DELETE, SHIFT-DELETE and RETURN) will overwrite the previous twelfth character. (I know that's hard to understand. Try it; you'll see what I mean).

The final general prompt is *(opt1, opt2, opt3)?* When this appears in the input window, **Debug+** wants you to press one of the keys separated by commas inside the parentheses. An example of this is *(D,P)?* This asks you what device you want to use, the disk or printer. Pressing D specifies disk, and so on.

Display formats

Debug+ has two memory display modes: disassembly and memory dump. A disassembled line looks like this:

```
ADDR OP B1 B1 xMNE
```

where *ADDR* is the address, *OP* is the op code, *B1* and *B2* are the operands, and *xMNE* is the mnemonics. The *x* indicates the direction of a relative instruction (branch instruction). If the arrow points up, it's a backwards branch; and if the arrow points down, it's a forward branch. A memory dumped line looks like this:

```
ADDR B1 B2 B3 B4 B5 B6 123456
```

where *ADDR* is the address, *B1-B6* are the values in the locations, and *1-6* are the ATASCII character representations of the values.

Commands

To use one of the commands in the table printed up above, just press the key on the keyboard which corresponds to the character listed under "key". For instance, to update the register display line, just press the *R* key.

Address set (*)

This is how you tell **Debug+** where your disassembly or memory dump will begin. After entering your address, **Debug+** will display the contents of the specified location.

Display toggle (D)

This command will flip the screen between disassembly and memory dump, and vice versa.

Quit DEBUG+ (Q)

This command returns you to DOS. To re-enter **Debug+** from DOS, do a RUN AT ADDRESS, with the address being Al00. To do this from OSS OS/A+, type *RUN A100* and press RETURN. From Atari DOS 2.0S, type *M*, hit RETURN, then type *Al00* and hit RETURN.

Go at address (G)

When this command is entered, the 6502 registers are loaded with the contents of the user registers, then program execution begins at the address specified. The user program will continue to execute until it is stopped by a 6502 BRK instruction, one of the break points, or by your pressing CTRL-ESC (the CONTROL and ESCape keys at the same time).

If one of these events occurs, your program will be interrupted, the 6502 registers will be saved in the user registers, and **Debug+** will take control. Then the location where the program was stopped will be disassembled or dumped on the screen, and the register line will be updated with the contents of the user registers.

Trace program (T)

When trace is activated, **Debug+** will ask for the starting address, then begin executing the code at the given location, one instruction at a time. You'll be asked to specify the speed of the trace in the prompt *(F,S,O)?* If you press *F*, then the trace will be as fast as possible; the *S* key will pause one-quarter of a second between each instruction; and the *O* key will cause the trace to be stepped by pressing the OPTION key.

A trace will be aborted if the tracer finds:

(1) a 6502 BRK instruction
(2) a break point
(3) an illegal instruction, and
(4) by pressing the ESC key

Note: if you're using the stepped mode, you must hold down OPTION and press ESC to abort. The trace can be paused by pressing the SPACE BAR.

The tracer does have some limitations. First, it cannot trace itself, so never try to trace **Debug+.** Second, any attempt to trace the real-time I/0 routines (such as disk or cassette access) will almost certainly fail. During a trace, the user register line will be updated before and after the execution of each instruction, so you can examine the register contents at any time.

Print disassembly (P)

When this command is executed, you're asked what device the output will go to (disk or printer) and what the starting address is. If disk output is chosen, a filename will be asked for, then disassembly will begin. During the process, the address just sent out to the device is displayed on the STAT line. To abort this command during execution, press any key.

If you choose the disk as the output device, **Debug+** will create a text file of the disassembly. You can then load this into a word processor or other text editor, and edit the disassembly.

Erase memory (E)

When this command is executed, you'll be asked for a starting and ending address of the erase. After this, the memory between and including those addresses will be erased. **Debug+** will fill all those addresses with zeroes (the 6502 BRK instruction). If the end address is smaller than the start address, you'll be given an ADDRESS RANGE ERROR, and the command will be aborted.

Change one byte of RAM (C)

When you enter this command, you'll be asked what address to change and what to change it to. At this point the contents of that memory location are changed. This is similar to the BASIC POKE statement.

Make sure that you don't change any of the memory that **Debug**+ uses. Also, careless choice of a change address may wipe out vital system data and cause a complete system lock-up. Take care.

Change register value (N)

When this command is entered, you'll be asked what register to change. To choose the register, simply press one of the following keys: A for Accumulator, X for X-Register, Y for Y Register, S for Stack Pointer, or P for Processor Status.

Next, you'll be asked the new value. Enter this, and the register contents will be changed. Notice that the register line on the screen has been updated to reflect the new value.

A note about the register line. If you examine the line on the screen, you'll see five small boxes and one large box, labeled *NV_BDIZC*. Under the label, there will be eight binary digits. This is the processor status, broken down into its flags. The following table tells you what each flag is:

N	=	Negative number flag
V	=	Overflow flag
_	=	Unused (always shown as set)
B	=	BRK instruction flag
D	=	Decimal flag
I	=	Interrupt flag
Z	=	Zero flag
C	=	Carry flag

If a 1 appears under one of these flags, it indicates a "set" or "true" condition. So, if the I bit is 1, then an interrupt is occurring.

Display registers (R)

All this command does is update the register line. It will display the current values of the user registers.

Set/Reset a break point (B)

A break point will stop the execution (or trace) of a user program. When you enter this command, a *B* will appear in the input window, indicating that you must either set or reset a break point.

Setting a break point is easy. Simply type the break point number (a single digit between 1 and 6), followed by a comma, then the address where you wish the break point to be. A typical line would look like this: *B1,0600[RETURN]*. This tells **Debug+** to set break point number 1 at location $0600 (or 1536 decimal). If break point number 1 is already set, you'll be given an error. If location $0600 contains a BRK instruction, you will also get an error. All 6502 BRK instructions are considered break points.

When you disassemble the address where a break point is set, the mnemonics will be shown in inverse video. This is how **Debug+** graphically shows the user where break points are. If you're in memory dump mode, the value for the address where the break point is set is also in inverse video.

Resetting a break point is easier than setting one. At the *B* prompt, simply enter the break point number you want to reset, followed by a RETURN. An example would be *B1 [RETURN]*. This will reset break point number 1. If it was not set, you'll get an error.

When a break point is reset, the old opcode is restored at the break point address, and the address for the break point is reset to $0000. If you look at the break point line at the bottom of the screen when you load **Debug+,** you'll notice that the addresses of all the break points are 0, and that there are six free break points.

Save a binary file (S)

This command allows you to save a single-stage binary file to disk. You'll be prompted to enter a filename, followed by the starting and ending addresses of the save.

Once again, if the ending address is less than the starting address, you'll be given an error. The file created with this function can be loaded from DOS. You cannot specify a run address for the file from **Debug+** , so you may want to append one to the file from BASIC or DOS.

Load a binary file (L)

This will load any binary file into memory. **Debug**+ will not run this file. After you enter the load filename, **Debug+** will load the file into memory, then the initial load address will be displayed on the STAT line.

Find a string in memory (F)

This command will locate all matches of any string -- up to twelve characters in length -- in memory. You'll be asked to enter the string you want to find and press RETURN. **Debug**

+ will clear the screen, and every match of what you typed in will be displayed on the screen, in this format:

```
FIND # nn Hexadr Decadr
```

where *nn* is the find number, and *Hexadr* and *Decadr* are the hex and decimal address of the find. The search will continue until the end of memory is reached (address $FFFF or 65535 decimal).

You may find it useful to begin the find at a specific address. To do this, terminate your string with a comma. **Debug+** will ask you what address to start from, and the search will begin as described above.

Debug+ has certain "reserved" characters that cannot be entered into your input and consequently can't be searched for. They are:

(1) RETURN (ATASCII 155)
(2) DELETE (ATASCII 126)
(3) SHIFT-DELETE or DELETE LINE (ATASCII 156)
(4) ESC (ATASCII 27)

All these characters are used by **Debug+** as either delimiters or cursor controls. However, all other control characters and alpha-numerics are at your disposal. One other thing: if you wish to search for a character that you usually generate by pressing ESC, then the key (such as clear screen -- CTRL-CLEAR), enter it without pressing ESC. So, to enter the clear screen character, just press CTRL-CLEAR.

High speed display (H)

This command will continually scroll the display through memory. You'll be asked what direction to scroll in -- up or down -- then the scrolling will begin. Use this command if you want to get somewhere very fast, since the screen will move at blinding speed. To pause the scroll, press SPACE; to abort scrolling, press ESC.

Decimal/Hex and Hex/Decimal conversions (.)

This command will convert one number base to the other and display the results on the STAT line. If a hexadecimal number is entered, a decimal number will be generated, and vice versa. After the number to be converted has been entered, the STAT line will look like this: *nnnn = xxxx,* where *nnnn* is what you typed, and *xxxx* is its converted form.

Scroll up in memory (-)

When you press the hyphen key, **Debug+** will move the display window up 1 byte in memory. If you're in memory dump, **Debug**+ will move the window up 6 bytes in memory.

You may notice that it will usually take several key-presses to scroll up one instruction in memory. This is because **Debug**+ has no idea where the previous instruction starts, or how many bytes long it is. When this command is executed, the addresses will get lower.

Scroll down in memory (=)

This command will move the display window down one full instruction in memory (or 6 bytes in dump mode). When scrolling down, **Debug+** knows where the next instruction starts. That's why it can move down one full instruction. When using this command, addresses in the display window will get higher.

<u>Notes</u>

(1) For every command discussed above, the keyboard auto-repeat is active. So, by holding one of the command keys down, you can re-execute the command as fast as a key will repeat. This is very useful while using the scroll commands.

(2) While executing any command, pressing the ESC key will abort that function. If you are entering a filename and decide not to load a file, just press ESC at any time, and the command will abort.

(3) To pause the screen scrolling at any time, press the SPACE BAR. To abort a command that constantly scrolls the screen, press ESC.

<u>Don'ts</u>

Well, that's about everything you can do with **Debug+**. Now let me tell you what you *can't* do.

(1) Do not alter any of the contents of the memory between $Al00 and $C0FF. This is where **Debug+** resides. Changing any of this memory could be fatal.

(2) **Debug+** uses the VBREAK vector located at address $0206-$0207 to detect the 6502 BRK instruction. Do not alter this vector. Doing so will severely cripple **Debug+**. When you exit **Debug+**, the program will restore the default VBREAK vector.

(3) **Debug+** also uses the VKEYBD vector located at address $0208-$0209 to detect the CTRL-ESC key combination during a user run. Do not alter this vector. If you do, you may never be able to stop the execution of one of your programs. When **Debug+** is exited, it restores the default vector.

(4) When you press RESET while in **Debug+**, the program will take control and reinitialize itself. Do not alter the DOSINI vector located at $0C-$0D. When you exit **Debug+**, the default DOSINI vector is restored.

Well, that's it. **Debug+** will run as advertised, unless you break one of the rules I've outlined above. It should prove to be an invaluable aid to you in debugging machine language programs.

[Bryan Schappel is a Computer Science major at the University of Wisconsin. He's been programming the Atari for three years. Besides writing programs, his main computer interests are word processing and data management.]