MON+
Symbolic Debugger

for the ProDOS 8 version of ORCA/M

By Dave Winzler of Microseeds, Inc.

The Byte Works Inc.

1010 1101 0110

MON+
6502/65CO2 Symbolic Debugger

MON+ is a powerful debugging tool for tile ORCA environment. It provides a complete set of memory examination and manipulation commands, arithmetic and logical functions, a disassembler, a miniassembler and, as its most powerful feature, a step and trace facility that includes the ability to set and clear break points as will as have free-running subroutines. The disassembler, the mini-assembler, and the tracer all support a symbol table. Entries can be either manually-defined, or included as data tables in the object code of the program being debugged, and copied automatically into the symbol table. Likewise, the standard 65CO2 instruction set is supported by these three functions, although the tracer will only work properly if there is a 65CO2 processor installed in the machine.

Each of the MON+ commands is explained in detail below. In the description that follows, the meta-symbol *adr* (address) stands for any hexadecimal number in the range $0000..$FFFF. A user-defined symbol (see SYMBOL below) may be used anywhere adr may be used, provided the symbol is preceded by a @ sign. The meta-symbol *byte* stands for any hexadecimal number in the range $00.41717. Square brackets around command parameters indicate that the bracketed parameter is optional.

The meta-symbol *filename* can be replaced by any valid ProDOS file name or full or partial path name.

Using MON+

Before using MON+, it must be installed in the utility subdirectory. This is the same directory that contains all of the ORCA utilities, such as PEEK. If, for example, you are using the default configuration, you would copy MON+ to the Utilities disk using the command

```
COPY /MONPLUS/DEBUG /UTILITY
```

The next step is to enable the DEBUG command using the ORCA COMMANDS utility. The reference manual for ORCA explains how to do this. Add DEBUG as a utility command. After installing MON+, you can execute it from any directory by typing `DEBUG`.

The program's version number and copyright notice will be displayed, and the cursor will appear next to the prompt (which is the + symbol). Any of the MON+ commands may then be entered; for example, to display memory in the range $2000 through $201F, enter the command:

```
        LIST 2000.201F
```

Most commands may be shortened to one or two characters; the above example could have been shortened to:

```
        L 2000.201F
```

The commands are checked in the order in which the HELP command lists them, and the first one which matches the entered command is used. Extra letters are ignored. The above example could be lengthened to:

```
        LISTLESSLY 2000.201F
```

and it would still have done the same thing.

Using Break Points

When the SETB commands is used to set a break point, MON+ stores the break point address away in its break point table, along with the operation code at that address, and replaces the operation code with a BRK instruction. When control passes to that address, the BRK instruction is treated like an interrupt; a routine in the Apple F8 Monitor ROM saves the contents of tile 6502 registers, and passes control to MON+'s break point handier (via the hook at $3FO). MON+ fixes the stack pointer, tells the user where the break occurred, and checks to see if that address is in its break point table. If it is, MON+ replaces the original operation code, executes the instruction, replaces the BRK instruction, and enters the trace mode. If the BRK instruction isn't in the break point table, MON+ will simply wait for the next command.

To use a break point, it's necessary to have both the program being debugged and MON+ in memory at the same time.

While debugging a program using break points, it's most convenient to have both an assembly listing of the program, as well as a link editor listing, which shows where each subroutine starts. It is then a simple matter to add the relative address within each routine to the absolute address of the start of the machine (using MON+'s ADD instruction) to get the absolute address of an instruction within a routine. Here's a quick and easy method for getting MON+ and a program into memory at the same time, with MON+ in control:

1) Type DEBUG.

2) Use MON+'s BLOAD command to load in the program.

3) At this point the program can be listed, break points set, symbols entered, and the program traced. All of MON+'s other features may be used to debug the program as well.

4) When a session is done, use the QUIT command to return to the ORCA monitor.

If it's more convenient, it isn't always necessary to have the most recent listing, although obviously the more recent the listing, the better. Use the FIND command to search for a series of bytes in the vicinity of a target area, and then use LIST to pinpoint the actual addresses or code in question.

Note that when a break point is encountered, the break point handler will execute the instruction and stop *after* the break point, not before. For example, suppose that in the following piece of code:

```
3A06: LDA #6
3A08: STA $3A74
3A0B: JMP $3A05
```

a break point is placed at the STA instruction at $3A08. When that break point is encountered, the STA instruction is executed, MON+ displays the instruction and the registers, enters the TRACE mode automatically, and then waits for a key press. At this point, since the STA instruction has been executed, location $3A74 contains a 6.

A more sophisticated way of dealing with break points - and one which can all but eliminate the need for a current program listing for reference - is to use symbols to refer to addresses; for an explanation of symbols, see the SYMBOL and GETSYM commands described below.

Memory Usage

MON+ loads into memory at $9000. Except for the bottom thirty-two bytes of the stack, ($100 - $11F), MON+ does not directly use any memory below $9000. It "borrows" page zero memory by swapping a copy of user page zero and MON+'s page zero.

Input and Output

Like most debuggers, MON+ uses the input and output routines provided by the operating system. This makes it difficult to trace screen output of keyboard input operations. As a result, you should always execute subroutines that will do screen output or keyboard input at full speed, using the GO or JUMP commands.

Disk input and output presents another problem. Disk routines are very time critical, and cannot be traced using a debugger. It is best to always execute calls to ProDOS at full speed, again using the GO command. Unless you over ride MON+ it will automatically execute any JSR to either ProDOS or the ORCA/HOST shell at full speed.

MON+ Commands

```
ADD   [adr1[adr2[adr3 ... ]]]
```

Displays the sum of the operands (adr1 + adr2) in hex and, in parentheses, decimal. Zero or more operands may be specified.

```
AND   [adr1[adr2[adr3 ... 13]
```

Displays the logical (bitwise) AND of the operands. Zero or more operands may be specified.

`ASM adr`

Enters the mini-assembler. The current address (set if adr is specified) is displayed, followed by a colon. The miniassembler is ready for input. A RETURN exits the mini-assembler.

If you enter something on the command line other than a valid 65CO2 instruction mnemonic, MON+ will try to use it as a command. For example, you can still list memory while assembling a program.

The mini-assembler's syntax is slightly different from the standard 6502 syntax in that no operands are specified for the accumulator addressing mode. ROR means rotate right accumulator, but ROR A means rotate right memory address $0A. All numbers in the mini-assembler are interpreted as hexadecimal, so the leading dollar sign ($) is optional. Symbols may be used wherever an address is expected, provided they have been previously defined and are preceded with the @ sign. All 65CO2 mnemonics and addressing modes are supported.

An example of the use of the ASM command, with symbols, is shown below. The symbol ABC must have been defined previously.

```
+ASM 2000
2000: LDA 5000
2003: STA @ABC
2006:
+
```

The mini-assembler was exited by typing RETURN in response to the 2006: prompt, causing the MON+ prompt to re-appear.

There are two important predefined symbols, @PRODOS and @ORCAHOST. These are the locations of the ProDOS entry vector ($BFOO) and the ORCA/HOST entry vector ($OOFD). MON+ handles these symbols in a special way. Whenever a JSR to one of the symbols is assembled, MON+ automatically leaves three bytes after the JSR to give you room to add the ProDOS or ORCA operation code and a pointer to the parameter table.

`BLOAD filename`

Loads the binary file specified by *filename*. The file is loaded at its default address. An error message is generated if *filename* isn't found, or if it's not a binary file. Needless to say, if a file is loaded on top of either MON+ ($9000 - $BFOO) the operating system ($D000 - $FFFF), or ORCA ($800 - $lFFF) the system will crash.

```
BRUN  filename
```

BLOADs *filename*, then does a JSR to the file's load address.

```
CONTINU
```

CONTINU is a special form of the JUMP command that continues execution from the last executed location. For example, after tracing the program with the TRACE command, you might CONTINU program execution at full speed.

```
CAT   [filename]
```

Displays a disk catalog. *Filename* is optional. If specified, it must be the name of a directory file; that directory will be cataloged.

```
CLRB  adr
```

Clears the break point specified at *adr* and displays a list of the remaining break points. Beeps if there is no break point at *adr*. If *adr* is not specified, the routine displays the current break points. If *adr* is specified, and there is a break point at *adr*, the routine removes the break point and restores the original operation code at *adr*.

```
CLRSYM
```

This command clears the symbol table, setting it back to its original state. It does not remove the two special symbols, @PRODOS and @ORCAHOST. Since the command is potentially dangerous, you will be asked if you are sure about the choice before the table is cleared.

```
COMPARE  adr1.adr2  adr3
```

Compare the contents of the memory range *adr1* through *adr2* with a range of the same length, starting at *adr3*. If any byte does not match, the address within the first range is displayed, along with the differing bytes. If any of the operands are missing, an error message is displayed.

```
CONFIG
```

CONFIG allows the selection of two operational "preferences." In response to the CONFIG command, two prompts are displayed:

```
        Display stack? N

        Pause on JSR? Y
```

The default responses are indicated in boldface; to accept them, simply press RETURN. The Display Stack option enables or disables the printing of the values of the top eight stack entries during tracing, causing the trace for each instruction to span two lines; by disabling this feature, only a single-line instruction disassembly and register dump is listed. The Pause on JSR? option enables or disables a trace. With pause enabled, rather than continuing directly to the next instruction, a key press must be made. At this point, the subroutine may be executed as "free-running" -- that is, not traced, by pressing the R key. See the description of the TRACE command for details.

```
EOR    [adr1[adr2[adr3 ... ]]]
```

Displays the logical (bitwise) exclusive OR of the operands. Zero or more operands may be specified.

```
FIND   adr1.adr2   list
```

Find and display all occurrences of list in the memory range *adr1* through *adr2*. *List* may be composed of hex bytes or ASCII strings within quotes; use double quotes for high order bit on, single quotes for high order bit off.

```
FILL   adr1.adr2   byte
```

Fills the memory range *adr1* through *adr2* with die value *byte*. An error message is generated if any operand is missing,. Because it is a potentially dangerous command, it responds with an 'Are you Sure' prompt. Enter Y to execute, N to cancel.

```
GO [adr]
```

Calls *adr* as a subroutine (performs a JSR to *adr*). If *adr* is not specified, GO uses the current valued of the program counter, stored at $3A and $3B.

When an RTS is encountered, GO returns control to the MON+ command handier.

```
GETSYM adr
```

Copies symbol table entries from memory (presumably, from a loaded code file containing them), starting at *adr*. The format of the symbol table entries is:

bytes 1..10	Label name, with high bit of each character set
bytes 11..12	Address, in low-high order

A list of entries may be placed in memory; the end of die list is indicated by a zero. The GETSYM command is intended to provide a simple yet effective mechanism for transferring symbolic values from a program into the MON+ symbol table. There is no direct mechanism for transferring all global labels from an object module into the MON+ symbol table; instead, labels whose value is desired for debugging a given subroutine or series of routines are manually entered into the program source code in the following way:

```
ABC    START
XYZ    GEQU   $1234
       LDA    #5
       STA    XYZ
       RTS
       END

SYMBOLS DATA
       DC     C'ABC  ',A'ABC'
       DC     C'XYZ  ',A'XYZ'
       DC     H'00'  INDICATES END OF TABLE
       END
```

If the address of SYMBOLS is known (from the global symbol table listing of the link editor - in this example, suppose the value is $2006), and the code file has been previously loaded into memory from MON+, a GETSYM command of the form

```
GET 2006
```

will copy the label names and their values into the MON+ symbol table.

Now if a TRACE command, for example, is used on the short program listed above, symbols will appear in disassembled trace:

```
2000: A905   ABC    LDA  #$05  A=05 X=00 Y=D8 P=30 S=83
2002: BD3412 STA    XYZ        A=05 X=00 Y~D8 P=30 S=83
2005: 60     RTS               A=05 X=00 Y=D8 P=30 S=83
```

Symbols may also be entered manually after a GET, and multiple GETs may be used, with the same or different address; if a label with the same address as an existing label is entered, a Duplicate Symbol error message is displayed, and the new entry is skipped. For manual symbol entry, see SYMBOL.

There is a macro file on the MON+ disk called M6502.MONPLUS. It contains a macro called SYM, which simplifies the inclusion of symbol tables in a source program. Its usage is:

```
       SYM    ABC
       SYM    XYZ
       DC     H'00'
```

This will create a symbol table identical to the one indicated above; note that a zero must still be placed at the end of the table.

MON+ starts out with two symbols already defined. ProDOS is defined as $BFOO, and ORCAHOST is defined as $OOFD. These special symbols cannot be deleted. They are marked in the symbol table with a special P flag.

```
HELP
```

Displays a listing of all of the available MON+ commands.

```
JUMP  (adr]
```

Transfers control to *adr*; essentially, does a JMP to *adr*. If *adr* is not specified, it uses the current value of the program counter, stored at $3A and $313. JUMP is useful for continuing a program in real time after a break point.

```
LIST adr | LIST adr1.adr2 | LIST | LIST adr ln
```

Disassembles a machine language program. If a range is specified, the entire memory range is disassembled. If *adr*, *adr1.adr2* or *ln* are not specified, the next twenty instructions are disassembled. If In is specified, the next In instructions are disassembled. 65CO2 operation codes will be disassembled when encountered. Disassembly of JSR instructions to either of the special labels (@PRODOS or @ORCAHOST) will display the operation code and parameter table address in a special format in the operand field.

```
LABEL [adr label]
```

LABEL is another name for the SYMBOL command. See the SYMBOL command for a description of its use.

```
MEM adr | MEM adr1.adr2
```

Displays contents of memory in both hexadecimal and ASCII. If only one address is specified, the contents of the address are displayed. If a range is specified, the contents of that memory range are displayed. It is an error if *adr* is missing. (Note the description of the shorthand notation described immediately below.)

```
MEM adr [list]
```

Store *[list]* into memory starting at *adr*. *[list]* can be composed of hexadecimal bytes or ASCII strings within quotes (double quotes for high order bit on, single quotes for high order bit off).

For example:

```
        MEM 2000 A9 'Z' 20 ED FD 'A long string'
```

There is a shorthand notation for either version of the MEM command: the arguments of the MEM command, such as a number range

```
        2000.20FF
```

may be entered without being preceded by an explicit MEM command; MON+ defaults to interpreting commands as MEM only if numeric arguments are provided.

`MOVE adr1.adr2 adr3`

Copy the memory range *adr1* through *adr2* to memory starting at *adr3*. The routine can move a block of memory up or down, with or without overlaps, with no side effects. An error message is displayed if any of the operands are missing. Because it is a potentially dangerous command, it responds with an 'Are you sure?' prompt. Enter Y to execute, N to cancel.

`NEGATE      adr`

Displays the two's complement (negative) of *adr*. Extra operands are ignored.

`NOT   adr`

The one's complement of the operand is displayed. Additional parameters are ignored.

`OR    [adr[adr2[adr3 ... ]]]`

Displays the logical OR of the operands. Zero or more operands may be specified.

`PAGE  [adr]`

Displays the contents of an entire memory page, starting at *adr*. If *adr* is missing, the routine continues displaying the next page.

`PREFIX filename`

Set the default prefix to *filename*.

`PROFF`

Turns off the printer. Subsequent output is sent to the screen.

`PRON`

Turns on the printer. Subsequent output is sent to the printer.

```
QUIT
```

Returns to the ORCA monitor via an RTS. It also restores tile BRK vector (if it was changed by SETB).

```
READ  blockl[.block2]   adr[,SSlot,Ddrive]
```

Reads 512 byte disk blocks starting with *blockl* and continuing to *block2* into memory starting at *adr*. Because the command is potentially dangerous, it responds with the 'Are you sure?' prompt.

```
REG
```

Displays the current 6502 register values (A, X, Y, Status and Stack Pointer). Their values may be changed as each register is displayed in turn. Press RETURN to leave the contents alone, or enter a new hex value to change the contents of the register being displayed.

```
SETB  [adr]        [label]
```

Sets a break point at adr and displays a list of all break points. If adr is missing, the routine simply displays the break points. If adr is specified, it is added to the list of active break points. The operation code that was at adr is replaced with a BRK instruction; it then modifies the BRK vector to point to the MON+ break point handler, after first saving the original value.

The optional label parameter allows a symbolic name to be assigned to the break point as it is defined.

SUB   [adr1[adr2[adr3

... I]]

Displays the difference of the operands (*adr1 - adr2 ...*). Zero or more operands are allowed. If more than two operands are supplied, tile second through tile last are all subtracted from the first.

```
SYMBOL  [adr label]
```

Associates *adr* with the symbol label. If *adr* is already in the symbol table replaces that entry with *label*, but leaves the break flag intact. If *label* is already assigned to another address, an error message is displayed and the command is ignored. If no operands are specified, the entire symbol table is displayed.

Once a symbol has been defined, it may be used anywhere an adr may be used, including within the mini-assembler. For example, if MYLAB is associated the address $2000 (SYM 2000 MYLAB), these two statements produce identical results:

```
LIST 2000
LIST @MYLAB
```

Or from the mini-assembler:

```
3000: LDA @MYLAB
```

or like so:

```
ADD 10 @MYLAB
```

The result is $2010.

`TRACE [adr]`

Executes (and traces) code starting at *adr*. If the operand is missing, it executes the instruction at the location of the current program counter stored at $3A and $313. When this command is invoked, the instruction a *adr* is executed, and the instruction mnemonic and the contents of the register are displayed. The following single-key commands are active during a trace:

SPACE	single steps one instruction, then pauses.
RETURN	traces quickly until a key is pressed.
F	causes a fast trace (turns off display) until and RTS is encountered.
R	executes a subroutine in real time (only available after a JSR is displayed; causes a fast-trace if another instruction is displayed.
C	continues the program in real time (no trace).

ESC	returns to the MON+ command handler.

```
VECTOR
```

Displays the current RESET, BRK and lRQ vectors. Their values may be changed as each vector is displayed in turn. Press RETURN to leave the contents alone, or enter a new hex value to change the contents of the register being displayed. (Use with caution!)

```
WRITE block1[.block2]  adr[,Sslot,Ddrive]
```

Writes the disk blocks starting with *block1* and continuing to *block2* from memory starting at *adr*. Optional slot and drive parameters may be specified. Because the command is potentially dangerous, it responds with the 'Are you sure?' prompt.

MON+ COMMAND SUMMARY

ADD	[adr1[adr2[adr3 ...]]]	addition: *adr1* + *adr2*
AND	[adr1[adr2[adr3 ...]]]	logical and: *adr1* AND *adr2*
ASM	adr	enters mini-assembler
BLOAD	filename	load binary file into memory
BRUN	filename	loads and runs binary file
CONTINU		continue program execution
CAT	filename	displays catalog
CLRB	adr	clear break point
CLRSYM		clear symbol table
COMPARE	adr1.adr2 adr3	compare memory
CONFIG		configure debugger
EOR	[adr1[adr2[adr3...]]]	exclusive or: *adr1* EOR *adr2*
FIND	adr1.adr2 list	find *list* in memory range *adr1-adr2*
FILL	adr1.adr2 byte	fill memory range *adr1-adr2* with *byte*
GO	[adr]	execute code at *adr* (via JSR)
GETSYM	adr	gets symbol table entries from memory
HELP		list commands
JUMP	[adr]	execute code at *adr* (via JMP)
LIST	adr1 [.adr2] [ln]	disassemble code
LABEL	[adr label]	define label; same as SYMBOL
MEM	adr1 [.adr2]	display or change memory byte (or range)
MOVE	adr1.adr2 adr3	copy memory range *adr1 -adr2* to *adr*
NEGATE	adr	two's complement: 0 MINUS *adr*
NOT	adr	one's complement: *adr* EOR #$FF
OR	[adr1[adr2[adr3...]]]	logical or: *adr1* OR *adr2*
PAGE	[adr]	display memory page
PREFIX	filename	set prefix
PROFF		turns off output to the printer
PRON		turns on output to the printer
QUIT		re-enters ORCA monitor
READ	blockl [.block2] adr[,S,D]	read disk blocks
REG		display/change 6502 registers
SETB	[adr] [label]	set break point
SUB	[adr1[adr2[adr3...]]]	subtraction: *adr1* - *adr2*
SYMBOL	[adr label]	associates *label* with *adr* in symbol table
TRACE	[adr]	trace program
VECTOR		display/change RESET, BREAK and IRQ vectors
WRITE	block1.block2] adr[,S,D]	write disk blocks