ABZmon, a monitor debugger, by Alain Birtz
version 0.9b1

650 Grand St-Charles,
P.Q., CANADA, J0E-1A0
CompuServe [72467,2770]

This is a beta release of ABZmon. I ran the monitor on a plain Mac Plus, and on the SE and IIx with many INITs without problem, but there are so many system configurations... Any comments and suggestions will be greatly appreciated. Please report any bug found (where and when the bug occurred). ABZmon can be distributed freely, but please, keep the 3 files together.

Thank You.

There are 3 files in the monitor folder

- ABZmon: The actual monitor, an INIT to put in the system folder

- SaveMon: Used to save the current monitor setting

- MonMan: This document file...

How to install the monitor

To install the monitor add the 'ABZmon' INIT in the System folder and restart the computer. If the monitor is correctly loaded you will see this icon at the bottom of the screen.

If the monitor is not correctly loaded you will see this icon:

As soon as you see the monitor icon, the monitor is active, and you can call the monitor immediately (for example, to check how other INIT are installed).

You can disable the loading process by keeping the 'Option' and = keys pressed at the startup.

Note: the icon symbol is the one found on your computer's interrupt switch...

Why ABZmon

Mainly because I love the Mac... and dislike the the MS-DOS program with command line interpeter. Mac is easy to use. And must be still easy to use. If you have ever use MacsBug, you have noted the regressive prompt string style horror!

MacsBug is a powerfull debugger, with plenty of usefull function, but it not a Mac program. Mac program uses mouse, window, menu, highlighted item, etc. My goal with ABZmon is to give to the programmer a more user friendly interface. Debugger is a kind of tool that must not interfere with any program. A way to do this is to not use any Toolbox routines. ABZmon has a build-in set of custom windows, menus, dialog box, and highlighted items. It is not the Mac interface, but it is close.. and you can use the mouse as for usual Mac application.

ABZmon is small: on a Mac Plus it takes only 77K, and this includes 21K to save the screen. This leaves about 55K to ABZmon itself. Half of this code is keeping by the disassembler. The other half is mainly to reproduce an 'easy to use' interface (about 20K).

ABZmon is free. Give it to your friends. The only restriction is to keep the 3 files together. ABZmon will be improved. The next thing I will do, is to use the internal program label, MMU and FPU opcode need more explicitness, and I must give some tool to investigate the heap.

Calling the monitor

There are many ways to enter in the monitor:

- The interrupt (programmer) key on the side of the Macintosh

- The PowerOn key (only on ADB keyboards) or PowerOn-Option key or PowerOn-Command key or PowerOn-Option-Command key

- The 'Command-Option-Delete' or 'Command-Option-Backspace' keys

- By including '_Debugger' trap call in your code. The monitor stops the program after the trap call.

- By including '_DebugStr' trap call in your code. A string pointer must be pushed in the stack. This string is shown in the message monitor window.

- By including '_DebugNum' trap call in your code (see below).

- On a bus error, address error, illegal instruction, etc...

Other special keys:

- Shift-PowerOn key or Command-Option-~ to refresh the screen. Use this when the desktop screen is corrupted.

- CapsLock-PowerOn key or Shift-Command-Option-Delete (Backspace) to perform an ExitToShell call. Use this when a program freezes.

- Control-Command-PowerOn key or Shift-Command-Option-~ to restart the computer

- Shift-Control-Command-PowerOn key to do a ShutDown

The previous keys are on both upper corners of the main keyboard pad. You can change these keys with the help of SaveMon application.

- Esc or Tab key to switch between the desktop screen and the monitor screen.

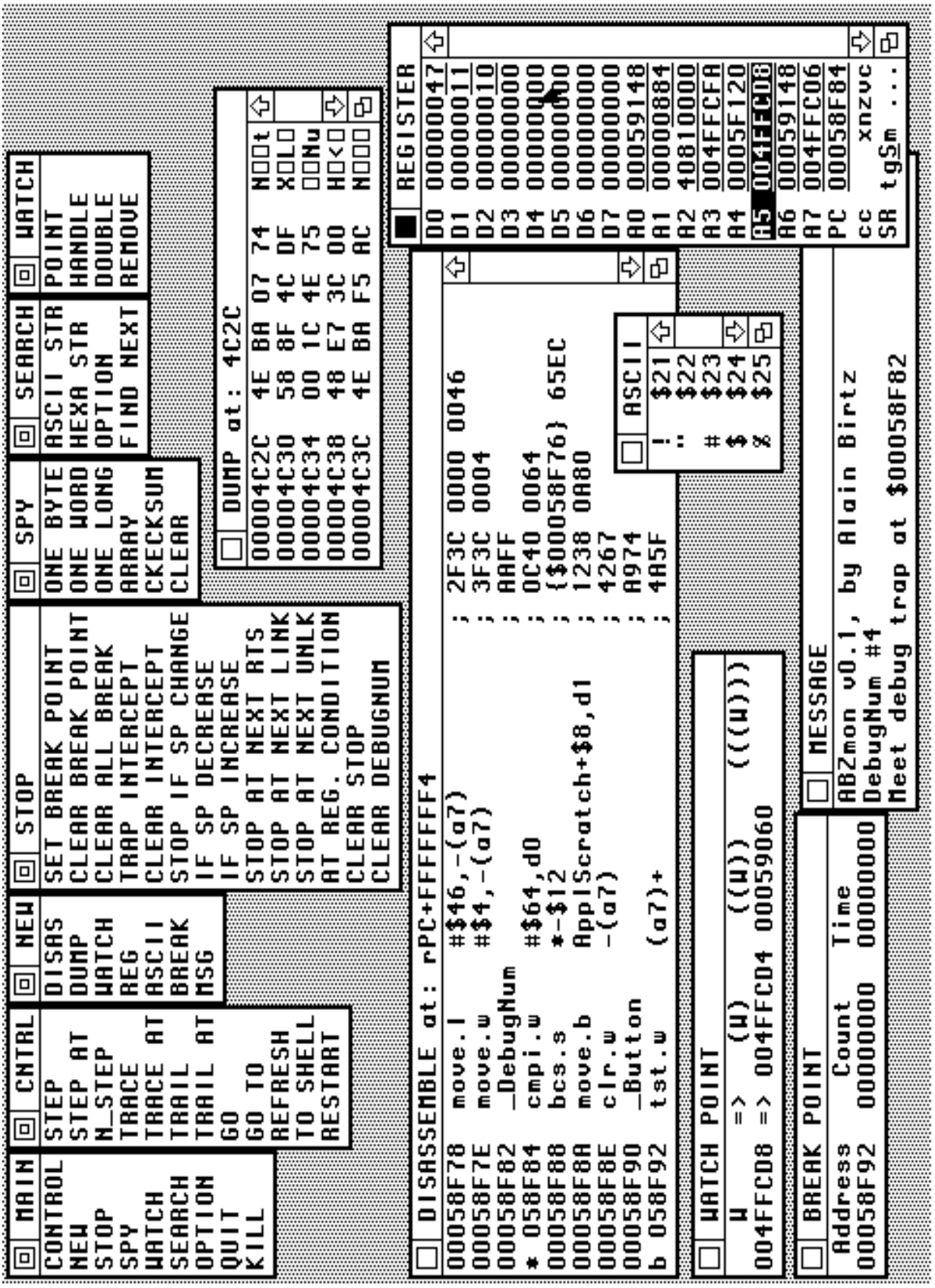
- Return, Enter or Space Bar to repeat step, trace or trail command.

- Command-w to close the active window, cmd+q to return.

- '.' Activates the previously opened window. Can be used to go through the window list.

- ',' Toggles between the last two active windows.

A typical monitor screen on the MacII



The monitor menus

The monitor menus sit on the top of the screen and are identified by a small square in the close box. Unlike the Mac menu, the monitor menus can be moved and closed (the first MAIN menu cannot be closed) and duplicated. To select an item in the menu just click the item as usual.

The MAIN menu

CONTROL: to open the CNTRL menu.

NEW: to open the NEW menu.

STOP: to open the STOP menu.

WATCH: to open the WATCH menu.

SEARCH: to open the SEARCH menu.

OPTION: to select form and type of disassembling.

QUIT: to quit the monitor, returning to the application/desktop.

KILL: the vectors modified by the monitor are restored, the monitor will no longer be active. **Use this carefully**.

The CNTRL menu

STEP: executes MC680X0 instructions, at current PC, one by one. All register, dump, watch windows are updated according to the affected register/value. Shortcut -> s

STEP AT: same as STEP but asks for the start address. Shortcut -> Shift s

N_STEP: same as STEP AT but asks for the number of instructions to execute before returning to the monitor.

TRACE: similar to STEP but BSR, JSR and LineA trap (OS or Toolbox trap call) are executed like a single instruction. The number of instructions executed are shown in the message window.
Shortcut -> t

TRACE AT: same as TRACE but asks for the start address. Shortcut -> Shift t

TRAIL: functionally the same as TRACE but instead of using the processor trace mode, a break point is set after each instruction. This increases the execution speed of BSR, JSR and LineA trap instructions. Shortcut -> l

TRAIL AT: same as TRAIL but asks for the start address. Shortcut -> Shift t

GO: to quit the monitor and return to the application/desktop. Shortcut -> g

GOTO: same as GO but asks for the starting address. Shortcut -> Shift t

REFRESH: quits the monitor and redraws the desktop screen. Useful to debug code using QuickDraw.

TO SHELL: performs an _ExitToShell before returning to the application/desktop.
Shortcut -> Shift-Command-Option-Delete (BackSpace) or CapsLock-PowerOn (only for ADB keyboard).

RESTART: restart the computer. Shortcut -> Shift-Command-Option-~ or Control-Command-PowerOn key (only for ADB keyboard).

The NEW menu

DISAS: asks for starting address and opens a disassemble window

DUMP: asks for starting address and opens a dump window

WATCH: opens a window showing all the watch point/handle.

REG: opens a window showing the D0...D7, A0...A7, PC and SR register values

ASCII: opens a window showing an ASCII chart.

BREAK: opens a window showing all the break points.

MESSAGE: opens a window showing the last 3 monitor messages.

The active window

This window has a black close box. The first time you click on a window (menu) it becomes active. You can move the active window as usual, the mouse being in the title window area. You can close the active window by clicking the close box. If the window has a scroll bar you can click in the arrow to scroll up and down or use the up and down key on the keyboard. if the window has a grow box you can use it to shrink or expand the window as usual

The default address

You can highlight a line in the active disassemble/dump/watch/register/break window by clicking this line. In the register window you must click in the value area (the register name area is reserved to change the register value). In the dump window you must click in the dump address area (the other part are reserved to change the memory).

The highlighted line sets the default address according to the value in this line. The default address is the one shown in many dialogs asking you for an address.

For example: to set a break point, highlight the address in some disassemble window and click on the SET BREAK POINT in the STOP menu. To execute the code starting at the address in the A3 register, highlight the A3 line in the register window and click the GO TO item in the CNTRL menu.

The start address

To open a new disassemble or dump window you must give the start address. The start address can be an hexadecimal number, a register value with optional offset or a register indirect value with two optional offsets. For example:

123ABC or $456def
rA6 the start address is E5F5E (see the monitor screen above)
rA6 the start address is E5F5E (see the monitor screen above)
rA6-$2 the start address is E5F5C
[a6+4]+6 if the address at E5F64 is D000 then the start address is D006

Register name and address can be entered with lower or upper case letter, a register name can be preceded by the letter 'r' or 'R' to indicate a register name. An hexadecimal value can be preceded by the symbol '$' to indicate an hexadecimal value. a6 refer to register A6 not to the $A6 hexadecimal value. The '*' symbol can be used as PC register name.

The dissassemble window in the monitor screen (above) has start address rPC.

The Dump window

There are 3 parts in each line. The first part holds the dump address. The second part shows the hexadecimal memory byte at this address and the third part shows the ASCII representation of the the hexadecimal memory byte. Clicking on the first part highlight the line and set the default address. Clicking on the second part show a dialog to set 4 consecutive bytes starting at the address of the byte under the cursor. Clicking the third part shows a dialog to set 8 consecutive characters starting at the address of the char under the cursor. The rectangle character in this area replaces the unprintable characters. In the dialog, the rectangle character has an ASCII value of 137 and not the value shown in the dump window, so the greatest care must be taken when you modify memory this way.

The disassemble window

Each line disassembles one instruction. The first part of each line is the address of the instruction, the second part is the symbolic instruction, the third part is the hexadecimal instruction code and, if the PC relative address is used, the absolute address is shown between braces. The trap instruction and the low memory global are identified by their name.

The PC instruction is identified by an asterisk (*) at the beginning of the line. A break point is identified by the letter b.

Note: all 68000, 68020, 68030 are disassembled except for the MMU code found in the 68030. The code for 68881/68882 are not yet implanted. However these coprocessor instruction are seen like cpBcc, cpBDcc, cpGEN and cpRESTORE.

The watch window

Shows the complete list of the watch point/handle/double-handle. The first column shows the point address, the other two show the indirection value.

The register window

The first part shows the register name and the second the register value. Clicking on the first part displays a dialog to change the register value. Clicking on the second part highlights the line and sets the default address.

The ASCII window

Shows an ASCII chart for characters between 0 and $7F.

The break window

Shows the complete list of break points. The first column shows the break point address, the second, the number of time requested before stopping, and the third, the number of time that the monitor meets this break point.

The message window

Displays the monitor last 3 messages.

The dialog window

This window appears at the center of the screen, has buttons and edits fields. The tab key can be used to switch between fields. The return key is always equivalent to the OK button. If an error occurs in some edit field (wrong number, invalid address...) the dialog is displayed again.

The stop menu

SET BREAK POINT: to set a break point and the number of times that the monitor meets this break point before stopping (0 -> first time, 1 -> pass one time then stop the next time, etc). The monitor stops when the PC is equal to the break point address, so the instruction is not executed. To execute the break point instruction use the STEP, TRACE, TRAIL or GO commands.

CLEAR BREAK POINT: clear a break point at the given address (as for many commands, you can highlight the break point address before calling this command, the address becomes the default address and you just need to hit the return key).

CLEAR ALL BREAK: remove all break point.

TRAP INTERCEPT: the monitor checks all calls to the lineA trap in the specified range (A000-AFFF or A000 AFFF or for single trap A100). Like for break points, you must set the number of times that the monitor meets these traps before stopping. The monitor stops when the PC is equal to the trap address, so the call is not executed.

CLEAR INTERCEPT: removes intercept process. You must use this command as soon as possible, to restore the normal processor speed.

STOP IF SP CHANGE: executes instructions step-by-step while the SP register does not change its value. This is a very slow process, floppy disk operations are agony... (but you can see, in slow motion, for example, how the System does a window redraw,).

IF SP DECREASE: same as ' STOP IF SP CHANGE' but stops only when SP decreases. This can be used to stop at the next JSR, BSR or to check when some parameters are pushed on the stack.

IF SP INCREASE: same as ' IF SP DECREASE' but stops only when SP increases. Useful to return from JSR or BSR.

STOP AT NEXT RTS:
STOP AT NEXT LINK:
STOP AT NEXT UNLK: to stop after or before procedure/function.

AT REG. CONDITION: you must give the register name, the condition and the value to compare with the register. For example D0>7. There are four conditions [=,<,>,<>] the last one being 'not equal'. The execution continues as long as the condition is not met.

CLEAR STOP: removes previous stop process. You must use this command as soon as possible, to restore the normal processor speed.

The SPY menu:

Executes instructions step-by-step while the specified memory location is not changed. Like for previous STOP item this is a very slow process.

ONE BYTE: check for a byte at the given address.
ONE WORD: check for a word at the given address.
ONE LONG: check for a long word at the given address.

ARRAY: the monitor keeps an image of this array in the spy buffer and after each instruction compares the array and the spy buffer. If some byte has changed the monitor stops the execution of the program. The default size for the spy buffer is 256, but it can be changed with the SaveMon application.

CHECKSUM: similar to array, but a checksum is made instead of a comparison byte for byte.

CLEAR: removes any spy process. You must use this command as soon as possible, to restore the normal processor speed.

The WATCH menu

POINT: set a watch point. You must give the point address. This address can be any ROM or RAM location. The long integer value at this point can be seen in any open watch window.

HANDLE: set a watch handle point. The long integer value at this point and the long integer value pointed by the first value can be seen in any opened watch window.

DOUBLE: same as HANDLE but with 2 degrees of indirection.

REMOVE: remove all watch points, closing any watch window.

The SEARCH menu

ASCII STR: search for an ASCII string in memory in the specified range. For example: search for Apple between 1A00 and FFFF (1A00-FFFF or 1A00 FFFF). You can use the wildcard symbol (? by default, but can be changed with the option item) to match any character. Search can be made case sensitive or not (changed by the option item also). For a large range, on the classic Mac, searching can take several seconds. When the monitor finds the string, it displays the address where the string was found. The DUMP button will automatically open a dump window at this address (however there is a possibility of maximum of 30 windows opening at the same time...).

HEXA STR: same as ' ASCII STR', but the string is given by hexadecimal number. For example 4A00?57

OPTION: to set a new wildcard, when using the wildcard and make search case sensitive or not.

FIND NEXT: to quickly find the next occurrence of the previous string.

The DebugNum trap

This call works like Debugger or DebugStr, but instead of stopping the program and entering in the monitor just after the call in the code, DebugNum can stop the program only after a given number of time. In fact, DebugNum takes 2 parameters. One parameter, called 'count', is a long integer. If count=0, the stop occurs the first time the program meets DebugNum, if count=1, the first DebugNum call is ignored and the stop occurs the next time the program meets DebugNum, etc. The other parameter, called 'Number', is a word integer from 0 to 9 that gives the DebugNum id so you can use up to ten different DebugNum calls.

The call from the assembly language looks like this

```
move.l  #70,-(sp)                ; count
move.w#4,-(sp)           ; DebugNum #
dc.w             $AAFF
```

or, if you use MPW assembler

```
move.l  #70,-(sp)                ; count
```

```
        move.w#4,-(sp)          ; DebugNum #
        _DebugNum
```

with the following inline of code in your header

```
_DebugNum     opword   $AAFF
```

For MPW C you must define in your header

```
void      DebugNum(count,number)        long number;    integer count; extern 0xAAFF;
```

and the call looks like

```
DebugNum(70L,4);
```

Resetting DebugNum

The monitor maintains an array of ten flags and ten long integers, one for each DebugNum call. When the monitor meets a DebugNum call for the first time, it sets the flag and fills the long integer with the value 'count'. If count=0 the monitor stops the program, otherwise the long integer is decreased by one. Each time the monitor meets DebugNum call, it checks the long integer. If the long integer is 0 the monitor stops the program, otherwise the long integer is decreased by one.

Suppose that in a debug process, your program exits before the long integer is 0. The next time you will use DebugNum (for the same 'number'), after you have recompiled you program for example, the monitor will initialize the long integer with the value 'count' only if the flag is cleared. You can clear one flag, or all flags with the help of the 'CLEAR DEBUGNUM' item in the 'STOP' monitor menu. So, each time you use DebugNum, make sure that the flags are clear.

KNOW BUGS

- When using any trace command (Step, spy, stop...), if the target code holds some instruction that store the SR (in stack or in memory), the SR will be stored with the trace bit set. So, when you later resume with the GO or similar command, a trace process will be induced (and the monitor will be called) when the SR is restored.

- The KILL command in the MAIN menu will not work correctly if some vector, used by the monitor, are changed after the monitor is loaded. For example you cannot run two debuggers at the same time.

The SaveMon application

When the monitor is correctly loaded you can use this application to save the current monitor window arrangement. You can also set the text cursor blinking rate and the spy buffer size (the current value is shown as default value). The 'Change Interrupt key' button can be used to change the 4 special keys to call the monitor, to perform an ExitToShell, a computer restart or a ShutDown. If you use the Cancel button, the current configuration will not be used.