Technote 1136

MicroBug: The ROM Debugger

By Louella Pizzuti
Revised by Quinn "The Eskimo!"
Apple Worldwide Developer Technical Support

CONTENTS

This Technote describes the features of MicroBug, the ROM debugger present in all Macintoshes from the Mac Plus onward.

This Note is directed at bored developers looking for something historically interesting to read on the train going home after a hard day's coding. It's also directed at developers trying to debug a problem that disappears when they install MacsBug.

About MicroBug

MicroBug was first introduced with the Mac Plus and has been a standard component in the ROM of all Mac OS computers since then. MicroBug is invoked when the system takes a Non-Maskable Interrupt (NMI) and there is no other debugger (e.g., MacsBug) installed.

MicroBug is not present on the Macintosh 128, Macintosh 512, and Macintosh XL computers.

Environment

MicroBug uses standard system services to operate its user interface. Specifically, MicroBug uses QuickDraw to draw its window and the Event Manager to get keyboard events. Internally, it shares a lot of code with the System Error Handler. This stands in contrast to MacsBug, which uses its own low-level drawing and input handling routines. The upshot is that MicroBug is less likely to be functional in the face of a "damaged" system than MacsBug.

Expressions

MicroBug expressions consist of:

hex number (with or without a leading "$")
: Yields the value of the hex number.

"."
: Yields the dot address. This address is initially set to 0 and is subsequently set by `DM` and `SM` commands.

@ <expr>
: Yields the value of the memory at <expr>.

RAx

Yields the contents of 680x0 address register Ax .

RDx
: Yields the contents of 680x0 address register Dx .

PC
: Yields the contents of the 680x0 Program Counter register.

- <expr>
: Yields <expr> negated.

<expr> + <expr>
: Yields the sum of the two expressions.

<expr> - <expr>
: Yields the difference of the two expressions.

All expressions have an inherent size, which is the number of valid bytes the expression returns. The size of a hex number is determined by the number of hex digits in the number. One or two hex digits yields a size of one (1) byte, three or four hex digits yields a size of two (2) bytes, and five or more hex digits yields a size of four (4) bytes.

The size of ".", indirect, and all register expressions is four (4) bytes.

The size of an arithmetic expression is the maximum of the size of the components.

MicroBug Commands

Dump Memory

```
DM addr

DM
```

With an argument, the dump memory command dumps the contents of memory at the address specified by `addr` . Without an argument, the dump memory command continues dumping memory from the memory location where the previous dump memory command left off.

Both variants set the dot address to the address where the dump started.

Typing Return after entering this command will continue dumping memory.

Set Memory

```
SM addr expr...
```

The set memory command sets the contents of memory, starting at the address specified by `addr`, to the values specified by the `expr` expressions. The command stores either a byte, word or long, depending on the size of the expression. On completion, the command displays a dump of the memory, starting at the first location set.

The command sets the dot address to the address where the dump started.

Typing Return after entering this command will continue dumping memory.

Go

G `addr`

G

With an argument, the go command starts execution at the address specified by `addr`. Without an argument, the go command resumes execution from where MicroBug was entered.

Total Display

TD

Displays all the standard `680x0` registers by dumping the memory where they were stored on entry to MicroBug. The following table describes the specific memory locations of interest:

Address	Size	Description	Name
`$0C30`	16 x 4 bytes	`680x0` registers d0-d7 and a0-a7 in that order	`SEVarBase`
`$0C6C`	4 bytes	`680x0` SP register	`MacsBugSP`
`$0C70`	4 bytes	`680x0` PC register	`MacsBugPC`
`$0C74`	2 bytes	`680x0` SR register	`MacsBugSR`

Typing Return after entering this command will repeat the command.

Set and Display Register

Ax `expr`

Ax

Dx `expr`

Dx

PC `expr`

PC

SR `expr`

SR

With an argument, these command set the specified register to the `expr` . Without an argument, these commands display the specified register. `x` determines the address or data register of interest, and can be any digit from 0 through 7.

Typing Return after entering one of these commands will repeat the command.

MicroBug Esoterica

Force Quitting

If you want to force quit the current process, you can just type:

```
SM 0 A9F4
G 0
```

This places an `_ExitToShell` trap at location 0 and then continues execution at that trap. Of course, this trick is now mostly pointless because Mac OS supports the Command-Option-Escape key sequence to force quit a hung process.

Some enterprising souls have discovered that commands like:

```
G FINDER
```

```
G EXITTOSHELL
```

will also force quit an application. This is because MicroBug's expression evaluator is extremely forgiving, so seemingly irrelevant strings will successfully evaluate to expressions. In both cases shown above, the resulting value is odd. Going to an odd address causes the 680x0 microprocessor (or the emulated microprocessor, on PowerPC-based computers) to take an "address error" exception. Mac OS responds to this exception by terminating the current process, which was the desired result.

Given the serendipitous nature of this feature, DTS continues to recommend the first sequence of commands.

Note:
A Pascal version of MicroBug's expression evaluation algorithm is shown below.

```
function StringToNumber(numberText : string) : longint;
    var
        result : longint;
        i : integer;
        digit : integer;
begin
    result := 0;
    for i := 1 to length(numberText) do begin
        digit := ord(numberText[i]);
        if digit > $39 then begin
            digit := digit - 7;
        end; (* if *)
        digit := band(digit, $000F);
        result := result * 16 + digit;
    end; (* for *)
    StringToNumber := result;
end; (* StringToNumber *)
```

Using this algorithm, the value for "FINDER" is `$00F27DEB` and the value for "EXITTOSHELL" is `$DD8C1E55`.

MicroBug and the Modem

If you crash into the debugger and the system hangs, try turning off your modem.

Note:
I have no idea whether this is still useful advice, but how could I remove such a cryptic comment from the technote? -- Quinn, 1998

Using MicroBug Seriously

If you're using MicroBug to debug a problem, it *really* pays off to have another machine running MacsBug right next to it. You can then use MacsBug commands to assist in your debugging. MacsBug commands like `dh` (to disassemble hex words), `mcd` and `wh` (to find low-memory globals), and `dm 0 <template>` (to find the offsets of fields within a structure) are all extremely helpful.

Summary

MacsBug is your friend. MicroBug is much less of your friend, but when you're a Mac programmer with a bug that disappears when you install MacsBug, you need all the friends you can get.

Further References

- *MacsBug Reference and Debugging Guide, for MacsBug version 6.2* , Apple Computer, Inc., Addison-Wesley, 1990

Downloadables

 Acrobat version of this Note (37K).

Change History

- Originally written in June 1986 as Technote 38 "ROM Debugger" by Louella Pizzuti.
- Revised in March 1998 to Technote PT 33 "The ROM Debugger" by Apple Developer Technical Support.
- Rewritten by Quinn "The Eskimo!" in August 1998 as Technote 1137 "MicroBug: The ROM Debugger".
  This version contains a more complete description of the debugger commands.
- Revised by Quinn "The Eskimo!" in September 1998 to explain why "G FINDER" works.

Acknowledgments

Thanks to Brian Bechtel and Jim Murphy. Special thanks to Clarus, for all those wacky Technote numbering schemes.