Using RomBug

Version 4.7
Copyright and publication information

This manual reflects version 4.7 of Microware OS-9. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.
## Contents

### Overview 7
- Using this Manual ........................................................................................................................................................................ 7
- Processor Specific Display Information ............................................................................................................................................... 7
- Invoking RomBug .................................................................................................................................................................................. 8
- Symbolic Debugging ............................................................................................................................................................................. 8
- Relocation Registers ............................................................................................................................................................................. 9
- Expressions .......................................................................................................................................................................................... 11
- Commands ........................................................................................................................................................................................ 13

### Commands 21
- Breakpoint ........................................................................................................................................................................................ 22
- Talk-through ......................................................................................................................................................................................... 22
- Download .......................................................................................................................................................................................... 23
- Execution ........................................................................................................................................................................................... 24
- Memory Change .................................................................................................................................................................................. 27
- Memory Disassembly .......................................................................................................................................................................... 29
- Memory Display .................................................................................................................................................................................. 30
  - Hex/ASCII Dump Memory Display ................................................................................................................................................. 31
  - Text Display ....................................................................................................................................................................................... 32
- Change Machine Registers .............................................................................................................................................................. 33
- Memory Fill ........................................................................................................................................................................................ 33
- Memory ............................................................................................................................................................................................ 35
- Search ............................................................................................................................................................................................... 35
- Memory Copy ..................................................................................................................................................................................... 39
- Linking to a Module ........................................................................................................................................................................... 40
- Symbolic Debugging ........................................................................................................................................................................ 41
- Attaching a Module .......................................................................................................................................................................... 43
- Viewing Expressions ........................................................................................................................................................................ 44
- OEMCMD ......................................................................................................................................................................................... 45

### 68xxx Processors 49
- -o Option ....................................................................................................................................................................................... 50
- Commands ........................................................................................................................................................................................ 50
- Supported Registers .......................................................................................................................................................................... 50
- Display Information ......................................................................................................................................................................... 54
- Change Machine Registers ........................................................................................................................................................... 64
  - Examples ........................................................................................................................................................................................................... 64
- Instruction Disassembly Memory Display ........................................................................................................................................ 66
- Floating Point Memory Display .................................................................................................................................................... 67
- Setting and Displaying Debug Options ......................................................................................................................................... 68

### Pentium and 80x86 Processors 71
- -o Options .................................................................................................................................................................................... 72
- Commands ........................................................................................................................................................................................ 73
MIPS Processors 137

-o Options.................................................................................................................................................................................. 138
Commands...................................................................................................................................................................................... 139
Supported Registers .............................................................................................................................................................................. 140
    General Purpose Registers.......................................................................................................................................................... 140
    Multiply and Divide Registers.................................................................................................................................................. 141
    Program Counter Register.......................................................................................................................................................... 141
    System Control Registers.......................................................................................................................................................... 141
    Floating Point General Purpose Registers................................................................................................................................ 143
Display Information............................................................................................................................................................................. 143
    Normal Register Display.......................................................................................................................................................... 143
    Status Register Display.......................................................................................................................................................... 144
    Floating Point Status and Control Register (FCR31).................................................................................................................. 146
Rombug Examples............................................................................................................................................................................... 147
    Setting Breakpoints............................................................................................................................................................ 147
    Trace Command............................................................................................................................................................. 147

SH-5 Processors 149

-o Options.................................................................................................................................................................................. 150
Commands...................................................................................................................................................................................... 152
Supported Registers .............................................................................................................................................................................. 152
    General Purpose Registers.......................................................................................................................................................... 153
    Program Counter Register.......................................................................................................................................................... 153
    System Control Registers.......................................................................................................................................................... 153
    Floating-point General Purpose Registers................................................................................................................................ 154
    Target Address Registers.......................................................................................................................................................... 155
Display Information............................................................................................................................................................................. 155
    Normal Register Display.......................................................................................................................................................... 155
    Status Register Display.......................................................................................................................................................... 156
    Floating-point Status and Control Register (FCR31).................................................................................................................. 157
Rombug Examples............................................................................................................................................................................... 157
    Setting Breakpoints............................................................................................................................................................ 157
    Trace Command............................................................................................................................................................. 158
Overview

RomBug is a privileged mode ROM debugger used to debug both system and user state programs. RomBug runs in supervisor state and takes control of the Central Processing Unit (CPU) when invoked.

RomBug uses an architecture named modular ROM which enables low level support configuration in a manner similar to the way OS-9® is configured. RomBug is configured as a low level module that gains access to the resources it needs by using other low level modules.

Among the low level modules used by RomBug is the debugger server. This module provides RomBug with the low level debugger services it needs such as the ability to do single step execution and to set breakpoints.

The RomBug client/server approach to low level debugging increases the ease of adding other debug clients to the system. This enables the use of more sophisticated debuggers, such as the Microware Hawk™ debugger tool.

The debugger command set allows analysis of programs by setting breakpoints, tracing control, and trapping exceptions. This can all be done symbolically. Extensive memory commands allow examination and changing of memory and register values and examination of CPU status/control registers.

Reference the Commands section in this chapter for a complete list of RomBug commands. Reference Chapter 2 Commands for details about RomBug commands.

The talk-through and download commands enable communication with the host system as a terminal and downloading of programs into RAM for testing via the communications link.

The debugger accepts command lines from the console in the form of a command code followed by pressing the [return] key. The backspace (<control>h) and line delete (<control>x) keys are used to correct errors.

Using this Manual

This manual describes RomBug operating under both OS-9 for 68K and OS-9. Where information provided in the manual is not applicable to both operating systems, the operating system to which the information is applicable is stated.

Processor Specific Display Information

RomBug is supported on various processors:

• ARM
• IBM PowerPC family
RomBug commands produce processor specific displays. Generally, the example displays in this manual, where common to all supported processors, reflect the 68k family of processors. Separate chapters in this manual are devoted to each processor family RomBug supports.

**Invoking RomBug**

All of the following methods call RomBug:

- ROM bootstrap
- Ultra C library function `_os_sysdbg()`
- the `break` utility

The kernel calls RomBug in system crash conditions. It may also be activated by any processor exception that the debugger is monitoring. For example, a hardware abort switch that causes an exception could invoke RomBug. Other ways RomBug may be activated are discussed later in this manual.

The first commands are usually commands to attach to all the symbol modules corresponding to the code modules to be debugged. Breakpoints are then set at the appropriate addresses and the `g` command is issued to return to normal timesharing. When the breakpoint is reached, control is returned to RomBug.

**Symbolic Debugging**

The RomBug symbolic debugging facility allows easy debugging without manually referencing external linkage maps or address tables for reference. The linker places the symbols associated with global code and data offsets into a symbol module. If a symbol module is available for the code module being debugged, symbolic addresses may be used in most debugger commands.

The `-g` option of the Ultra C linker creates a symbol module. The name of the module is that of the code module with `.stb` appended. If a directory named `STB` exists in the execution directory, the linker places the symbol module in that directory. This helps to minimize the number of entries in the execution directory.

When using the `a` or `am` command, the debugger looks for the symbol module in memory. If the symbol module is not located, the following message displays:

- can't link to module "prog.stb"

where `prog` is the name of the program module of interest.

When the symbol module is found, it is examined to verify that it matches the code module being debugged. The module Cyclic Redundancy Check Character (CRC) of
the code module is stored in the symbol module. If this CRC value does not match that of the code module, the following message displays:

- symbol module ‘prog.stb’ is obsolete, use anyway (Y/N)?.

Certain utilities (fixmod, etc.) change the module CRC that causes the above message to display. In such cases, the symbol module is correct and may be used regardless of the warning message.

This message indicates one of two problems:

1. The symbol module does not match the code module or the debugger does not recognize the format of the symbol module. The cause of the error is usually that an old version of the symbol module and/or code module is already in memory or the modules were modified by the fixmod utility, changing user-id, attribute, revision, etc. Ensure that the linker properly linked the program and that old versions of the program and/or symbol modules are removed from memory.

2. If the debugger cannot locate a symbol module, the program may still be debugged but the symbolic facilities are not available. Because the program’s symbols are kept in a separate module, a program does not require a final production compilation to remove the symbol information. Symbol modules of production programs may be retained should additional debugging be required.

RomBug must have gotten control while in the context of the module being debugged to accurately display (since it uses the current Global Data Register to calculate the absolute addresses) the location of data symbols. Code symbols should always be valid.

**Relocation Registers**

Relocation debugger maintains eight relocation registers. These registers are useful for storing memory base addresses for later use in commands and expressions. The relocation registers are referenced two different ways dependent upon the processor. For the 80x86 and 68k processors, relocation registers are referenced by the names r0 through r7. For the MIPS, PowerPC and SuperH processors, relocation registers are referenced by the names rr0 through rr7.

Relocation register 0 is hard-wired to zero. Whenever an address is specified, the default relocation register is added to the address automatically. Setting the default relocation register to zero disables this action. The default relocation register is not added if a symbolic address or an expression is specified. Relocation register commands are shown in Table 1-1. Relocation Register Commands.
Relocation registers are considered symbols by RomBug and are present in displays.

Table 1-1. Relocation Register Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Print default relocation</td>
</tr>
<tr>
<td>@&lt;num&gt;</td>
<td>Set default relocation register to &lt;num&gt;. &lt;num&gt; = 0 to 7.</td>
</tr>
<tr>
<td>.r</td>
<td>Display relocation registers.</td>
</tr>
<tr>
<td>.rr</td>
<td>Set specified relocation register to &lt;val&gt;</td>
</tr>
<tr>
<td>.r&lt;num&gt; &lt;val&gt;</td>
<td>Set specified relocation register to &lt;val&gt;</td>
</tr>
<tr>
<td>.rr&lt;num&gt; &lt;val&gt;</td>
<td>NOTE: Relocation register 0 is hard-wired to zero.</td>
</tr>
</tbody>
</table>

† .r for 68K and X86 processors and .rr for all others.

Examples

RomBug: @
the default relocation register is .r0 00000000
RomBug: .r4 1fe00
RomBug: @4
RomBug: @
the default relocation register is .r4 0001fe00
RomBug: .r

rn:00000000 00000000 00000000 00000000 0001fe00 00000000 00000000 00000000

/* using relocatable registers as a symbol
 - All command line addresses will be biased by the 
   value of the default relocatable register. If that 
   register's value is zero(0) then it is not displayed
*/
RomBug: v .r4 /* find current value of .rr4 */
0x0400F620 (67171872) 0x00000000+rr4
RomBug: @4 /* set .r4 as default relocation register */
RomBug: di 100 4 /* explicit addresses will be biased by .r4 */
0x00000100+rr4>0EC08CE0 add r12,r12,lr
0x00000104+rr4>407D86E2 add r7,r6,#0x1000
0x00000108+rr4>207B97E5 ldr r7,[r7,#0xB20]
0x0000010C+rr4>247097E5 ldr r7,[r7,#0x24]
dis: @0 /* set relocatable register to .r0 */
RomBug: di 100 4 /* address not biased */
0x00000100 >00000000 nop
0x00000104 >00000000 nop
Expressions

Any debugger command accepting an address or numeric value can also accept an expression. An expression operand consists of the elements identified in Table 1-2. Expression Operand Elements. Unary, binary, and indirect operators are identified in Table 1-3. Unary Operators (operate on the right operand), Table 1-4. Binary Operators (operate on the left and right operand), and Table 1-5. Indirect Operators respectively.

Table 1-2. Expression Operand Elements

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;symbol&gt;</td>
<td>Code or data symbol</td>
</tr>
<tr>
<td>&lt;num&gt;</td>
<td>&lt;num&gt; is interpreted as a number in the default radix</td>
</tr>
<tr>
<td>#&lt;num&gt;</td>
<td>&lt;num&gt; is a valid decimal (base 10) number</td>
</tr>
<tr>
<td>0x&lt;num&gt;</td>
<td>&lt;num&gt; is a valid hexadecimal (base 16) number</td>
</tr>
</tbody>
</table>
Using RomBug

Table 1-2. Expression Operand Elements

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;char&gt;</td>
<td>The ASCII value of &lt;char&gt; is sign extended to a 32-bit word.</td>
</tr>
<tr>
<td>&lt;reg&gt;</td>
<td>Valid register. For a complete list of machine registers used by supported processors, refer to the appropriate processor-specific chapter of this manual.</td>
</tr>
</tbody>
</table>

Table 1-3. Unary Operators (operate on the right operand)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e1</td>
<td>Negate e1</td>
</tr>
<tr>
<td>~e1</td>
<td>Complement e1</td>
</tr>
</tbody>
</table>

Table 1-4. Binary Operators (operate on the left and right operand)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1 + e2</td>
<td>Add e2 to e1</td>
</tr>
<tr>
<td>e1 - e2</td>
<td>Subtract e2 from e1</td>
</tr>
<tr>
<td>e1 * e2</td>
<td>Multiply e1 by e2</td>
</tr>
<tr>
<td>e1 / e2</td>
<td>Divide e1 by e2</td>
</tr>
<tr>
<td>e1 &gt; e2</td>
<td>Bitwise right shift e1 by e2 bits</td>
</tr>
<tr>
<td>e1 &lt; e2</td>
<td>Bitwise left shift e1 by e2 bits</td>
</tr>
<tr>
<td>e1 &amp; e2</td>
<td>Bitwise AND of e1 and e2</td>
</tr>
<tr>
<td>e1</td>
<td>e2</td>
</tr>
<tr>
<td>e1 ^ e2</td>
<td>Bitwise exclusive OR of e1 and e2</td>
</tr>
</tbody>
</table>
Chapter 1: Overview

All expression evaluation is performed using 32-bit two’s complement arithmetic. Traditional operator precedence is not observed; evaluation is left to right. Parenthesis (()) may be used to force evaluation order. Most commands requiring a count accept an asterisk (*) to mean infinity.

Commands

RomBug provides commands shown in Table 1-6.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Print Help</td>
</tr>
<tr>
<td>@</td>
<td>Print default relocation</td>
</tr>
<tr>
<td>@&lt;num&gt;</td>
<td>Set default relocation register to &lt;num&gt;. &lt;num&gt; = 0 to 7.</td>
</tr>
<tr>
<td>a</td>
<td>Attach to all modules found (after the system is up)</td>
</tr>
<tr>
<td>a [&lt;mod&gt;]</td>
<td>Attach to symbol module(s) for &lt;mod&gt;(s)</td>
</tr>
<tr>
<td>am&lt;beg&gt;&lt;end&gt;</td>
<td>Attach to all modules found in address range &lt;beg&gt; to &lt;end&gt;</td>
</tr>
<tr>
<td>b</td>
<td>List breakpoints</td>
</tr>
</tbody>
</table>
### Table 1-6. Commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>b</strong> &lt;addr&gt;</td>
<td>Set breakpoint at &lt;addr&gt;</td>
</tr>
</tbody>
</table>
| **c**[<size>] [n]<addr> | Enters the specified change mode at the address specified by <addr>. Data displays in big-endian format. <size> identifies the size of data to change (default is byte):  
1: change long word lengths  
w: change word lengths  
b: change byte lengths  
o: change byte lengths at odd addresses  
e: change byte lengths at even addresses  
n: specifies no echo when changing.  
RomBug displays the memory at the specified address and prompts for a new value. The change prompt controls are:  
+ move to next location  
- move to previous location  
<CR> move to next location and display memory  
<num> store new value and move to next address  
. exit change mode |
| **d**[<size>] [s] [<num>] [<expr>] [<len>] | Switch to dump memory mode. Displays memory in hex and ASCII. <size> identifies the size of data to change (default is byte):  
1: display long word lengths  
w: display word lengths  
b: display byte lengths  
s: specifies swap endianess for word and long access  
<len>: display <count> bytes from address <expr> (default is 256 bytes).  
<num>: (digit 0-9) how many lines to show if count is missing, any other command exits display mode  
In display memory mode, pressing [return] displays the same number of lines as previously displayed. Any other command exits display mode. |
Table 1-6. Commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd[&lt;num&gt;] [&lt;expr&gt;] [&lt;len&gt;]</td>
<td>Double precision floating point display mode. Display memory at &lt;expr&gt; for &lt;len&gt; bytes as double precision floating point. If &lt;len&gt; is unspecified, &lt;num&gt; specifies the number of lines to display (0-9). In display memory mode, pressing [return] displays the same number of lines as previously displayed. Any other command exits display mode. NOTE: Command not available on all processors.</td>
</tr>
<tr>
<td>df[&lt;num&gt;] [&lt;expr&gt;] [&lt;len&gt;]</td>
<td>Single precision floating point display mode. Display memory at &lt;expr&gt; for &lt;len&gt; bytes as single precision floating point. If &lt;len&gt; is unspecified, &lt;num&gt; specifies the number of lines to display (0-9). In display memory mode, pressing [return] displays the same number of lines as previously displayed. Any other command exits display mode. NOTE: Command not available on all processors.</td>
</tr>
<tr>
<td>di[&lt;num&gt;] [&lt;expr&gt;] [&lt;len&gt;]</td>
<td>Instruction disassembly mode. Disassemble instructions at &lt;expr&gt; for &lt;len&gt; instructions. If &lt;len&gt; is unspecified, &lt;num&gt; specifies the number of lines to display (0-9). In display memory mode, pressing [return] displays the same number of lines as previously displayed. Any other command exits display mode.</td>
</tr>
<tr>
<td>dn[e] &lt;cmd&gt;</td>
<td>Download S-Record code. e echos S-Records. If e is not specified, load addresses are displayed every 512 bytes. &lt;cmd&gt; is issued to the shell to trigger the download. The I/O delay must be set in relocation register 1 before the download.</td>
</tr>
</tbody>
</table>
### Table 1-6. Commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dx[num] [expr] [len]</td>
<td>Extended precision floating point display mode. Display memory at expr for len bytes as extended precision floating point. If len is unspecified, num specifies the number of lines to display (0-9). In display memory mode, pressing [return] displays the same number of lines as previously displayed. Any other command exits display mode. NOTE: Display type x performs the same as display type d for the PowerPC processor. NOTE: Command not available on all processors.</td>
</tr>
<tr>
<td>e</td>
<td>Enable/disable monitoring of processor-specific default exception vectors (system state only)</td>
</tr>
<tr>
<td>g</td>
<td>Go at Program Counter (PC)</td>
</tr>
<tr>
<td>g &lt;addr&gt;</td>
<td>Go at &lt;addr&gt;</td>
</tr>
<tr>
<td>gb</td>
<td>Go with boot staging calls to the debugger</td>
</tr>
<tr>
<td>gs</td>
<td>Execute next instruction and stop</td>
</tr>
<tr>
<td>gs &lt;addr&gt;</td>
<td>Execute until PC == &lt;addr&gt;</td>
</tr>
<tr>
<td>k &lt;addr&gt;</td>
<td>Kill breakpoint at &lt;addr&gt;</td>
</tr>
<tr>
<td>k*</td>
<td>Kill all breakpoints</td>
</tr>
<tr>
<td>l &lt;module&gt;</td>
<td>Link to memory module. Place address in relocation register 7.</td>
</tr>
<tr>
<td>mc&lt;dest&gt;&lt;src&gt;&lt;size&gt;</td>
<td>Copies memory from the address specified by &lt;src&gt; to the address specified by &lt;dest&gt;. The number of bytes to copy is specified by &lt;size&gt;.</td>
</tr>
<tr>
<td>mf[s] [n] &lt;beg&gt; &lt;end&gt; &lt;value&gt;</td>
<td>Fill memory range with the pattern in &lt;value&gt;. s specifies the size of the fill pattern (b, w, l). If s is not specified, a one byte pattern length is assumed. n indicates a non-aligned fill. Specify the address range with &lt;beg&gt; and &lt;end&gt; addresses.</td>
</tr>
</tbody>
</table>
### Table 1-6. Commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| `ms[<s>] [n] <beg> <end> [:<mask>] <value or string>` | Search memory range for `<value or string>` pattern.  
NOTE: Strings must begin with a quotation mark (").  
`s` specifies the size of the search pattern (b, w, l).  
If `s` is not specified, a byte pattern is assumed.  
`n` indicates a non-aligned search.  
The address range is specified by `<beg>` and `<end>` addresses.  
`:<mask>` is a bitmask applied to the search value. |
| `o<option>`      | Controls the various RomBug `<option>`s:  
`b<n>` Numeric input base radix  
`r` Toggle ROM type (soft) or RAM type (hard)  
`breakpoints (soft breakpoints not available on all processors)`  
`s` Toggle general register display  
`v` Display vectors being monitored  
`v` Display all exception vector values  
`v [-] [s | u] [d] <n> [<m>]`  
Monitor exception vector where:  
- To restore vector  
`s` System state only  
`u` User state only  
`d` Display only  
`<n>` Vector number in hexadecimal  
`<m>` Upper limit vector number in hexadecimal  
For processor-specific options, refer to the appropriate processor-specific chapter of the manual. |
<p>| <code>r [&lt;module&gt;]</code>  | Remove symbols for <code>&lt;module&gt;</code> |
| <code>r*</code>            | Remove all symbols |
| <code>rst</code>           | Reset the system |
| <code>s [mod:]&lt;symb&gt;</code> | Displays a single symbol from the current symbol module or symbol module <code>mod:</code>. The <code>*</code> and <code>?</code> wildcard symbols may be used in the symbol name. |
| <code>sc &lt;module&gt;</code>   | Shows code symbols for symbol module |</p>
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd &lt;module&gt;</td>
<td>Shows data symbols for symbol module (addresses based on current Global Data Register)</td>
</tr>
<tr>
<td>sm</td>
<td>Show symbol module directory</td>
</tr>
<tr>
<td>ss &lt;addr&gt;</td>
<td>Set default symbol module to module containing &lt;addr&gt;</td>
</tr>
<tr>
<td>ss &lt;name&gt;</td>
<td>Set default symbol module to module &lt;name&gt;</td>
</tr>
<tr>
<td>t [&lt;num&gt;]</td>
<td>Trace one or &lt;num&gt; instructions and switch to trace mode. Pressing [return] causes RomBug to trace another instruction in trace mode. Any other command causes RomBug to exit trace mode.</td>
</tr>
<tr>
<td>tm&lt;char&gt;</td>
<td>Talk mode. Escape from talk mode with &lt;char&gt;.</td>
</tr>
<tr>
<td>v &lt;expr&gt;</td>
<td>Print the value of the expression &lt;expr&gt; in hex and decimal</td>
</tr>
<tr>
<td>w[&lt;num&gt;]</td>
<td>Print subroutine stack; &lt;num&gt; specifies the number of calls displayed</td>
</tr>
<tr>
<td>x</td>
<td>External OEM command</td>
</tr>
<tr>
<td>xc [&lt;pid&gt;]</td>
<td>Display current [or &lt;pid&gt;] process information</td>
</tr>
<tr>
<td>xf</td>
<td>Display free memory information (similar to mfree -e)</td>
</tr>
<tr>
<td>xm</td>
<td>Display root module directory (similar to mdir -e)</td>
</tr>
<tr>
<td>xp [a]</td>
<td>Display processes, a = display alternate data (similar to procs -e[a])</td>
</tr>
<tr>
<td>xq</td>
<td>Display process queue information</td>
</tr>
<tr>
<td>xs</td>
<td>Display OS-9 system globals</td>
</tr>
<tr>
<td>xw &lt;addr&gt;</td>
<td>Display module that contains specified address</td>
</tr>
<tr>
<td>x?</td>
<td>Display OEMCMD help</td>
</tr>
</tbody>
</table>
Table 1-6. Commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.&lt;reg&gt; &lt;num&gt;</code></td>
<td>Set register <code>&lt;reg&gt;</code> to <code>&lt;num&gt;</code></td>
</tr>
<tr>
<td></td>
<td>NOTE: Relocation register 0 is hard-wired to zero. Reference the appropriate processor-specific chapter for processor-specific register information.</td>
</tr>
<tr>
<td><code>.r</code></td>
<td>† † Print relocation registers.</td>
</tr>
<tr>
<td><code>.rr</code></td>
<td>† † Print relocation registers.</td>
</tr>
<tr>
<td><code>.r&lt;num&gt; &lt;val&gt;</code></td>
<td>† † Set specified relocation register to <code>&lt;val&gt;</code></td>
</tr>
<tr>
<td><code>.rr&lt;num&gt; &lt;val&gt;</code></td>
<td>† † Set specified relocation register to <code>&lt;val&gt;</code></td>
</tr>
<tr>
<td></td>
<td>NOTE: Relocation register 0 is hard-wired to zero.</td>
</tr>
<tr>
<td><code>,</code></td>
<td>Print floating point registers</td>
</tr>
</tbody>
</table>

† Functionality available only for non-68K processors. OEMCMD must be initialized before using.
†† `.r` for 68K and x86 processors and `.rr` for all others.
2 Commands
Breakpoint

The debugger allows setting up to 16 simultaneous breakpoint addresses. Breakpoints must be set on word or long word addresses depending upon the CPU type.

Breakpoint commands are shown in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Display breakpoint list</td>
</tr>
<tr>
<td>b&lt;addr&gt;</td>
<td>Set breakpoint at the address specified by &lt;expr&gt;</td>
</tr>
<tr>
<td>k&lt;addr&gt;</td>
<td>Kill breakpoint at the address specified by &lt;addr&gt;</td>
</tr>
<tr>
<td>k *</td>
<td>Kill all breakpoints</td>
</tr>
</tbody>
</table>

Examples

RomBug: b
breakpoint count = 0
RomBug: b main
RomBug: b
breakpoint count = 1
main (00162f40)
RomBug: b main+1f0
RomBug: b
breakpoint count = 2
main (00162f40)
main+1f0 (00163130)
RomBug: k main+1f0
RomBug: k *
clear all breakpoints? y

Talk-through

RomBug uses the second serial port on the system for download and talk-through functions. Connecting this port to a host system effectively makes the target system terminal act as a host system terminal. The target system terminal may be used to edit, assemble, etc., on the host system, eliminating the need for two terminals.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tm &lt;EscChar&gt;</td>
<td>Enter talk-through mode. This mode is exited when the specified escape character is typed.</td>
</tr>
</tbody>
</table>
The escape character should be carefully selected to avoid duplication with the character used in normal communications with the host. For this reason, infrequently used characters such as the tilde (~) are recommended.

**Download**

The download command passes a command to the host system that causes it to send program data to the target system via the communications link. The program is loaded into RAM.

The program must be in the industry-standard Motorola S-record format. Only S1, S2, S3, S7, S8, and S9 record formats are recognized. The binex utility must be used to convert the Ultra C/C++ linker output from its normal binary format to S-record format.

The S-record format has data records that include a **load address** specifying where the program is to be loaded in memory. OS-9 for 68K/OS-9 programs are position-independent so the load address always starts at address zero. As S-records are received, the load addresses are added to the debugger default relocation register value to determine the actual address in RAM where the program is stored.

All program modules must be downloaded before executing OS-9 for 68K/OS-9 to enable the kernel module search to find them.

The relocation register must be set to the area of RAM reserved for downloaded code in the boot.a, a special search table. There are two versions of the download command as identified in Table 2-3. Download Command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dl &lt;HostCmd&gt;</td>
<td>Downloads data in Motorola S-record format. &lt;HostCmd&gt; is sent to host as a command line to trigger the download. I/O delay must be set in relocation register 1 before the download and address offset in the default relocation register. The load addresses are displayed every 512 bytes.</td>
</tr>
<tr>
<td>dle &lt;HostCmd&gt;</td>
<td>Same as dl &lt;HostCmd&gt; except received S-records are also displayed on the console instead of load addresses.</td>
</tr>
</tbody>
</table>

The <HostCmd> sent to the host is the command required to dump the S-record file. Ensure that the screen pause is turned off (using the tmode nopause command).

A sample download command is:
```
dl binex objs/boot320
```

On Unix, a sample command is:
```
dl cat s.rec.file
```
The debugger transmits the command string to the host and then expects host transmission of S-records. The download ends when an S7, S8, or S9 type record is received.

Sometimes the target system cannot keep up with a sustained high data rate when downloading. Therefore, the debugger sends XON and XOFF to the host for flow control. If the host system does not respond to XOFF immediately, a buffering delay count must be set up in relocation register 1 before using the download command (a value of 20 works well in most cases with a data link running at 9600 baud). Experimentation with this value may be required as it is dependent upon a combination of characteristics of the host system XOFF response lag time, the target system CPU speed, and the baud rate.

If the download command seems to hang up, the download may be aborted with a <control>e. This may also send a <control>e (abort signal) to the host system if the I/O buffer delay is not large enough or if the host’s screen pause is on.

- When the console port address is the same as the communication port address, the <control>e abort signal is ignored.

Downloading using these commands should only be attempted after a hardware reset or after a debugger rst command to prevent occurrence of stack/data conflicts within the OS and erroneous results.

When debugging just one module, keep it in a different file than the main OS-9 for 68K/OS-9 download file. When downloading a revised version considerable time is saved by downloading only the new version and using the main OS-9 for 68K/OS-9 code already in memory (use the rst command first).

To symbolically debug, add the .stb modules made by the linker to the download file. When debugging, use the am command to search the download area of memory for symbol modules.

**Execution**

RomBug provides the commands shown in the following table to initiate and control program execution.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Go. Continues execution at the current PC until a RomBug monitored event is encountered.</td>
</tr>
<tr>
<td>g &lt;addr&gt;</td>
<td>Go from address. Continues execution at &lt;addr&gt; until a RomBug monitored event is encountered.</td>
</tr>
</tbody>
</table>
Chapter 2: Commands

Table 2-4. Program Execution Commands (Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gb</td>
<td>Go boot. Stops at next stage of the OS-9 for 68K/OS-9 boot sequence. The first gb command goes until the bootfile is loaded. The next gb command goes until the module directory is built and the module CRCs are verified. The next gb command goes until the system is fully brought up.</td>
</tr>
<tr>
<td>gs</td>
<td>Go and stop. Continues execution at the current PC until the next instruction is encountered. This is the same as the g command but it sets a breakpoint at the next instruction. The breakpoint is automatically removed when the debugger regains control.</td>
</tr>
<tr>
<td>gs &lt;addr&gt;</td>
<td>Go and stop at address. Continues execution starting at the address in the PC register up to the specified &lt;addr&gt;. This is the same as the g command but it sets a breakpoint at &lt;addr&gt;. The breakpoint is automatically removed when the debugger regains control.</td>
</tr>
<tr>
<td>t</td>
<td>Trace. Traces one instruction and re-displays the machine registers.</td>
</tr>
<tr>
<td>t &lt;count&gt;</td>
<td>Trace instructions. Traces &lt;count&gt; instructions and re-displays the machine registers. Each instruction is displayed as it is executed. Breakpoints are ignored while tracing.</td>
</tr>
</tbody>
</table>

Examples

RomBug: g 

initial RomBug prompt after reset

BOOTING PROCEDURES AVAILABLE -- <INPUT> 

boot menu displayed

Boot from VME320 floppy drive - <f320>
Boot from VME320 hard drive -- <h320>
Boot from VME319 floppy drive - <f319>
Boot from VME319 hard drive --- <h319>

Restart the system ------------ <q>

Select a boot method from the above menu: abort switch pressed

<Aborted>

<Aborted>

Go and boot the system 

hard disk VME320 boot selected from above menu
A valid OS-9 for 68K bootfile was found.

RomBug has read bootfile into memory do not set breakpoints in any modules in the boot yet.

CRC has not been verified.

RomBug: gb

module directory is built and CRCs verified breakpoints may be set in OS-9 modules now

the startup file is read (and echoed)

Abort switch pressed.

Abort switch pressed.
Memory Change

The debugger memory change command is used to examine and change memory. When using this command, the debugger automatically enters the memory change mode. The following table identifies the memory change command forms.

Table 2-5. Memory Change Command Forms

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| c[<size>]<mode><addr> | Change memory values starting at the address specified by <addr>. The memory value type to be changed is specified by <size>. If <size> is not specified, byte values are assumed. <size> may be any of the following:

- b = change byte values
- o = change byte values at odd addresses
- e = change byte values at even addresses
- w = change word values
- l = change longword values
- q = change quadword values (64-bit processors only)

The change <mode> is specified as:

- n = Do not display value at address (does not read value)

Address boundaries must be consistent with the requirements of the processor in use.

The debugger displays the specified address and the value at the address in big-endian format, and prompts for the new value. After you enter the new value, the debugger displays the next address and its value and prompts again for a new value.

At the prompt, the following may be entered without changing any values:

- `Moves to the previous address and sets <cr> to next mode`
--- Moves to the previous address and sets <cr> to next mode
+ Moves to the next address and sets <cr> to last mode
= sets <cr> to hold mode to display the same address
<cr> Moves to the next, last, or same address and displays value
. Exits change mode and returns to the RomBug prompt
= <addr> Moves to an absolute address
n Toggle no-read
q Switch to changing quadword values
l Switch to changing longword values
w Switch to changing word values
y Switch to changing byte values

OS-9 Examples

RomBug: d1 .d0 display memory
0x1EF7E - 6A626364 65660000 00000000 00000000 jbcdef........
dis: c .d0 enter change mode
0x1EF7E :6A 'a store the character 'a'
0x1EF7F :62 20 store a blank
0x1EF80 :63 - back up
0x1EF7F :20 #20 different base (use 10)
0x1EF80 :63 - back up
0x1EF7F :14 + advance
0x1EF80 :63 . exit change mode
RomBug: d1 .d0 display memory
0x1EF7E - 61146364 65660000 00000000 00000000 a.cdef........

Changing longword values is similar.

tra: d1 0x1EF84 display memory
0x1EF84 - 00000000 00000000 00000000 00000000 ...............dis: cl 0x1EF84 enter longword change mode
0x1EF84 :00000000 #10000 new value
0x1EF88 :00000000 #44 new value
0x1EF8C :00000000 . exit change mode
RomBug: d1 0x1EF84 display memory
0x1EF84 - 00002710 0000002C 00000000 00000000 ..‘.........

The change values may also be given all at once with no intermediate memory display.

tra: d1 0x1EF84
0x1EF84 - 00000000 00000000 00000000 00000000 .................dis: cl 0x1EF84 #10000 #44 . enter change mode, store 2 values, exit change
RomBug: d1 0x1EF84
0x1EF84 - 00002710 0000002C 00000000 00000000 ..‘.....H........

OS-9 Example

RomBug: d1 .d3
$00020000 - 4F6F6F68 42616279 4F6F6F68 42616279 OoohBabyOoohBaby
dis: c .d3
Memory Disassembly

Memory is disassembled and displayed using the memory disassembly command, `di`, described in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>di &lt;addr&gt; [&lt;lines&gt;]</code></td>
<td>Disassembles and displays the specified number of machine instructions <code>&lt;lines&gt;</code> of memory starting at the address specified by <code>&lt;addr&gt;</code>. If <code>&lt;lines&gt;</code> is not specified, 16 lines of machine instructions are disassembled and displayed. <code>&lt;addr&gt;</code> must evaluate to an address that conforms to the processor-specific instruction alignment.</td>
</tr>
</tbody>
</table>

In the instruction disassembly display format, conditional instructions are sometimes followed with an “->” indicator. If -> is present, it indicates that the instruction performs its TRUE operation, otherwise the instruction performs the FALSE operation. The appropriate current condition code register is examined to determine which case the processor will perform (for instance, the -> indicator is based on the value of the condition register when the disassembly occurred).

- Each processor supported by OS-9 uses different condition code registers. Refer to the appropriate processor-specific chapter of the manual for a complete discussion on both the supported condition code registers and the conditional instructions.

### Table 2-6. di Command

- `$00020000 4F: 'a`
- `$00020001 6F: 20`
- `$00020002 6F: -`
- `$00020001 20: #20`
- `$00020002 6F: .`
- `RomBug: d1 .d3`
- `$00020000 - 61146F68 42616279 4F6F6F68 42616279 a.ohBabyOoohBaby`
- `dis: c1 .d3`
- `$00000000 - 00002710 0000002C 4F6F6F68 42616279 ..'....,OoohBaby`
- `dis: cl .d3 #10000 #44`
- `RomBug: d1 .d3`
- `$00020000 - 61146F68 42616279 4F6F6F68 42616279 a.ohBabyOoohBaby`
- `dis: d1 .d3`
- `$00020000 61146F68: #10000`
- `$00020004 42616279: #44`
- `$00020008 4F6F6F68 .`
- `RomBug: d1 .d3`
- `$00020000 - 00000002C 4F6F6F68 42616279 ..'....,OoohBaby`
- `dis: cl .d3 #10000 #44 .`
- `RomBug: d1 .d3`
- `$00020000 - 00002710 00000002C 4F6F6F68 42616279 ..'....,OoohBaby`
- `dis:`
Examples

RomBug: di main
main >4E550000  link.w a5,$0
main+54 >48E7CCA0  movem.l d0-d1/d4-d5/a0/a2,-(a7)
main+58 >41EE0E72  lea.l rest_env(a6),a0
main+5C >2008  move.l a0,d0
main+5E >61FF0000B36  bsr.l setjmp
main+514 >2A00  move.l d0,d5
main+516 >6710  beq.b main+$28
main+518 >7201  moveq.l #$1,d1
main+51A >2005  move.l d5,d0
main+51C >61FF0000B2BE  bsr.l put_exception
main+522 >95CA  suba.l a2,a2
main+524 >600000DA  bra.w main+$100
main+528 >61FF0000B2C2  bsr.l get_vectors
main+52E >42AE0E06  clr.l call_debug(a6)
main+532 >95CA  suba.l a2,a2
main+534 >61FF0000B5B4  bsr.l ConsSet

Memory Display

The `d` memory display command displays memory. This command allows memory to be displayed in a variety of ways.

> Endianness is preserved in a display memory command.

<table>
<thead>
<tr>
<th>Table 2-7. d Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
</tr>
<tr>
<td>d[{&lt;size&gt;[s]</td>
</tr>
<tr>
<td>&lt;addr&gt; [len]</td>
</tr>
</tbody>
</table>

By default, 256 bytes of memory are displayed in a normal hexadecimal/ASCII dump format. This can be further customized by using one or more `<size>` specifiers. `<size>` identifies the size of data to change (default is byte):

- l: display long word lengths
- w: display word lengths
- b: display byte lengths

In addition, l and w may be used with an `s` qualifier to cause the display to swap endianness for word and long access. Valid combinations:
Chapter 2: Commands

- **d**: default hexadecimal/ASCII dump format
- **db**: same as **d**
- **dl**: grouped longs
- **dls**: grouped swapped longs
- **dw**: grouped words
- **dws**: grouped swapped words

For display of memory as floating point, specify one of the optional format indicators **<M>**:

- **f**: single precision floating point format (four byte default display)
- **d**: double precision floating point format (eight byte default display)
- **x**: extended precision floating point format — only if Floating Point Unit is available (CPU specific default display)

If a format indicator is specified, the default number of bytes displayed changes in the following manner: 16 bytes for instruction disassembly and 4, 8, and CPU-specific for **f**, **d**, and **x** floating point formats, respectively. Use the **<len>** parameter to display a different number of bytes. **<len>** is the number of bytes to display and is rounded up in order to display a full line. For example, if 1 is specified, 16 bytes are actually displayed.

Another method for controlling the length of output is allowed by specifying the number of lines (0-9) to display: **<num>**. This value is used only if **<len>** is not specified.

For processors that do not support extended format, double format (**dd**) is used when extended format (**dx**) is specified.

**Hex/ASCII Dump Memory Display.**

In the ASCII field of the hex/ASCII dump, bytes in the range of **$20 - $7E** are displayed as the ASCII character equivalent. All other values are displayed as a period (**.**).
Using RomBug

Example Floating Point Memory Displays

dis: df 20200
$00020200 - 40490FDB 3.141592741012573

dis: dd 20000
$00020000 - 400921FB54442D18 3.141592653589793

dis: dx 20100
$00020100 - 40000000C90FDA22168C235 3.141592653589793

To display a floating point number in a machine register, specify an & followed by the name of the register. This will display the value in the register. Without the & the value in the register is interpreted as a pointer to the desired value.

Text Display

To display the contents of memory as ASCII text, use the dt command. It has all the same options as the d command, except that the size is always byte(b). The end-of-line character for the memory can be either CR, LF, or CRLF.

Example Text Memory Display

dis: 1 dbglog_mod
dis: dc .rr7+88 200
0x00000088+rr7:
ohci_root_ctrl_start: usb_transfer_complete(xfer=8fe5b4d8) status=0
ohci_root_ctrl_start: usb_transfer_complete(xfer=8fe5b4d8) status=0
ohci_root_ctrl_start: usb_transfer_complete(xfer=8fe5b4d8) status=0
ohci_root_ctrl_start: usb_transfer_complete(xfer=8fe5b4d8) status=0
ohci_root_ctrl_start: usb_transfer_complete(xfer=8fe5b4d8) status=0
ohci_root_ctrl_start: usb_transfer_complete(xfer=8fe5b4d8) status=0

0x00000000C90FDA22168C235 3.141592653589793

Example Floating Point Memory Displays

dis: df 20200
$00020200 - 40490FDB 3.141592741012573

dis: dd 20000
$00020000 - 400921FB54442D18 3.141592653589793

dis: dx 20100
$00020100 - 40000000C90FDA22168C235 3.141592653589793

To display a floating point number in a machine register, specify an & followed by the name of the register. This will display the value in the register. Without the & the value in the register is interpreted as a pointer to the desired value.

Text Display

To display the contents of memory as ASCII text, use the dt command. It has all the same options as the d command, except that the size is always byte(b). The end-of-line character for the memory can be either CR, LF, or CRLF.
Change Machine Registers

Use the dot (.) command, identified in the following table, to change the machine registers.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.&lt;reg&gt; &lt;val&gt;</td>
<td>Changes the specified register to &lt;val&gt;. If a floating point register is specified, the change value may be either a double precision decimal constant or a left-justified hexadecimal value: .&lt;fprev&gt; &lt;float-decimal constant&gt; or .&lt;fprev&gt; &lt;left-justified hex constant&gt; or .&lt;fprev&gt; .&lt;fprev&gt; 0x&lt;hexdigits&gt;</td>
</tr>
</tbody>
</table>

Changes to registers appear in the register display but do not actually occur until execution is resumed (with the g or t command)

Memory Fill

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mf [&lt;s&gt;] [n] &lt;start&gt; &lt;end&gt; &lt;value&gt; [&lt;value&gt;]</td>
<td>The mf command is used to fill memory with a given pattern. The fill pattern size is specified by the &lt;s&gt; parameter. &lt;s&gt; may be b, w, or l for byte, word, or longword, respectively. If &lt;s&gt; is not specified, a byte length is assumed. n indicates that the fill is to be performed without regard to word/longword boundaries (word and longword fills are done on a byte for byte basis).</td>
</tr>
</tbody>
</table>
<start> and <end> are the starting and ending addresses for the memory fill. <value> is the pattern used to fill the memory range. If the length of the fill determined from <start> and <end> is not an even word or longword multiple (for a word and longword fill), the length is trimmed to the next lowest respective multiple. <value> is the pattern used to fill the memory range. There are two special types of memory fill when using the byte fill size:

- <value> may start with a quotation mark ("). In this case, all remaining characters are used as a fill string.
- Multiple byte <value>s can be specified. In this case, each successive value is used as a fill character.

In both cases, the pattern is reused from the beginning if the fill count has not been exhausted.

Examples

RomBug: d3 70000 display memory
0x00070000 - FFEECODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070010 - FFEECODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070020 - FFEECODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@

dis: mfb 70000 70003 al fill with byte
RomBug: d2 70000
0x00070000 - A1A1A1A1 FEEDCODE FEEDCODE FFEECODE !!!!~m@~m@~m@~m@
0x00070010 - FFEECODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@

dis: mfw 70000 7000f 5252 fill with word
RomBug: d2 70000
0x00070000 - 52525252 52525252 52525252 52525252 RRRRRRRRRRRRRRR
0x00070010 - FFEECODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@

dis: mf1 70000 7000f 81186226 fill with longword
RomBug: d3 70000
0x00070000 - 81186226 81186226 81186226 81186226 ....b&..b&..b&..b&
0x00070010 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070020 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@

dis: mfwn 70001 70007 0102 non-aligned fill word
RomBug: d3 70000
0x00070000 - 81010201 02010201 81186226 81186226 ..........b&..b&
0x00070010 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070020 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@

dis: mfb 70000 7000f "shazam! fill with a string
RomBug: d4 70000
0x00070000 - 7368617A 616D2173 68617A61 6D21736B shazam!shazam!sh
0x00070010 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070020 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070030 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070040 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070050 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070060 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@
0x00070070 - FEEDCODE FEEDCODE FEEDCODE FEEDCODE ....m@~m@~m@~m@

The [n] parameter must be used on processors with word/longword boundary limits if the fill address does not conform to these boundary requirements.
Memroy

Table 2-10. ms Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| ms[<s>][n] <beg> <end> [:<mask>] <value or string> | Search memory range for <value or string> pattern.  
NOTE: Strings must begin with a quotation mark (".). The size of the search value is specified by the <s> parameter. <s> may be b, w, or l for byte, word or longword, respectively. If <s> is not specified, a byte length is assumed.  
n indicates that the search is to be performed without regard to word/longword boundaries (e.g., word and longword searches are done on a byte for byte basis). The address range is specified by <beg> and <end> addresses.  
: <mask> is a bitmask applied to the search value. |

Search

The [n] parameter must be used on processors with word/longword boundary limits if the fill address does not conform to these boundary requirements.

<start> and <end> are the starting and ending addresses for the memory search. If the length of the search is not a multiple of an even word or longword (for a word and longword search), the length is trimmed to the next lowest respective multiple.

<value> is the pattern used to search the memory range.

There are two special types of memory search when using the byte search size:

1. <value> may start with a quotation mark (".). In this case, all remaining characters are used as a search string.
2. Multiple byte <value>s can be specified. In this case, each successive value is used as a search pattern.
A `<mask>` may be specified to limit the comparison to only those bits set in the mask. If `<mask>` is not specified, the mask used is -1 (all bits set). The mask parameter is ignored for multiple character patterns.
Chapter 2: Commands

Examples

RomBug: dl btext
btext: - 4AFC0001 000259E 000C006A 00000048 J]....%....d...H

dis: msw btext btext+259e 1 search for word-aligned 0001

RomBug: msw btext btext+259e 4e40 search for system calls

_packchec+0x30 - 4B40008C 321F4E75 202E8000 90AB8008 N@..2.Nu

_trapinit+0x1A - 4E400021 64066100 12B6E666 2F490014 N@..d.a..Ned/I.

_getstat+0x2C - 4E40009D 65000294 02E2206F N@..e..j ..'..b


_dup+0x4 - 4E40008E 60000108 4E756080 2041222F N@..d..p4.N@..2.

_readln+0xA - 4E40009B 60DC48E7 60802041 22F00F010 N@..\".A/.


RomBug: `msl btext btext+259e .ffffff80 4e400000 only non-I/O calls`  
`sbrk+0x14` - 4B400007 64000008 225F6000 00642D40 N@.d."...d-@  
`_argmem+0x2` - 4B400002 650A2D40 84B0200A 245F4E75 N@.(e.-@.`_Nu  
`_artmem+0x8` - 4B400009 245F6000 00691C8 C1886406 N@.)$...HA.d.  
`_exit+0x2` - 4B400004 DEADDEAD 003C0001 4E754E75 N@..^..<..NuNu

RomBug: `ms ln btext btext+259e :ffffff80 4e400000 non-longword boundary search`  
`trapinit+0x1A` - 4E400021 64066100 12CE6564 2F490014 N@.!d.a..Ned/I..  
`trapinit+0xBA` - 4E400006 4E400006 12D866FC 4E7548E7 N@...Xf|NuHg  
`trapinit+0xBE` - 4E400006 12D866FC 4E7548E7 C080203C N@...Xf|NuHg@.<  
`ebrk+0x42` - 4E400028 204A245F 650000C8 2D4884D8 N@.( J$_e..H-X...  
`sbrk+0x14` - 4E400007 64000008 225F6000 00642D40 N@..d..."_'..d-@  
`_srqmem+0x2` - 4E400028 650A2D40 84E0200A 245F4E75 N@.(e.-@.' .$_Nu  
`_srtmem+0x8` - 4E400029 245F6000 000691C8 C1886406 N@.)$_'....HA.d.  
`_exit+0x2` - 4E400006 DEADDEAD 003C0001 4E754E75 N@..^-^-.<..NuNu

RomBug: `ms btext btext+259e *math string search`  
`trapinit+0x7B` - 6D617468 00000000 00223C00 0000402F math.."..../@/  
`trapinit+0x34` - 2A2A2A2A 20537461 63626572 666C6F77 * Stack Overflow

RomBug: `ms btext btext+259e same as previous but null terminated`  
`fclose+0xD` - 2A000C48 C0080000 0F671430 2A000C48 *.H@....g.0*..H  
`fclose+0x19` - 2A000C48 C0080000 01670E20 0A612628 *.H@....g...a(k

RomBug: `ms btext btext+259e 2a 00 same as previous but null terminated`  
`fclose+0x3B` - 2A00047EE 803E2004 720FC081 2C002200 *.Gn.>.x..",.  
`putc+0x7` - 2A0000C8 00280000 0802020C 08000880 *.H@....".....  
`putc+0x1B` - 2A0000C8 C07222C0 810C8000 00002266 *.H@...@......f  
`putc+0x35` - 2A0000C8 C0880000 02672A48 78000141 *.H@...g*Xx.A  
`putc+0x6B` - 2A0000C8 C0880000 08660620 0A610001 *.H@...f..a.  
`putc+0x8D` - 2A0000C8 C0880000 0766160C 97000000 *.H@..f......
Memory Copy

Memory is copied from one address to another using the memory copy command: mc. The copy memory command syntax is shown in the following table.

Table 2-11. mc Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc&lt;dest&gt;&lt;src&gt; &lt;size&gt;</td>
<td>Copies memory from the address specified by &lt;src&gt; to the address specified by &lt;dest&gt;. The number of bytes to copy is specified by &lt;size&gt;.</td>
</tr>
</tbody>
</table>

Examples

RomBug: d 100000

```
$00100000 - 01010101 01010101 01010101 01010101 .................
$00100010 - 01010101 01010101 01010101 01010101 .................
$00100020 - 01010101 01010101 01010101 01010101 .................
$00100030 - 01010101 01010101 01010101 01010101 .................
$00100040 - 01010101 01010101 01010101 01010101 .................
$00100050 - 01010101 01010101 01010101 01010101 .................
$00100060 - 01010101 01010101 01010101 01010101 .................
$00100070 - 01010101 01010101 01010101 01010101 .................
$00100080 - 01010101 01010101 01010101 01010101 .................
$00100090 - 01010101 01010101 01010101 01010101 .................
$001000A0 - 01010101 01010101 01010101 01010101 .................
$001000B0 - 01010101 01010101 01010101 01010101 .................
$001000C0 - 01010101 01010101 01010101 01010101 .................
$001000D0 - 01010101 01010101 01010101 01010101 .................
$001000E0 - 01010101 01010101 01010101 01010101 .................
$001000F0 - 01010101 01010101 01010101 01010101 .................
```

dis: d 200000

```
$00200000 - 00000000 00000000 00000000 00000000 .................
$00200010 - 00000000 00000000 00000000 00000000 .................
$00200020 - 00000000 00000000 00000000 00000000 .................
$00200030 - 00000000 00000000 00000000 00000000 .................
$00200040 - 00000000 00000000 00000000 00000000 .................
$00200050 - 00000000 00000000 00000000 00000000 .................
$00200060 - 00000000 00000000 00000000 00000000 .................
$00200070 - 00000000 00000000 00000000 00000000 .................
$00200080 - 00000000 00000000 00000000 00000000 .................
$00200090 - 00000000 00000000 00000000 00000000 .................
```

dis: mc 200000 100000 9f

RomBug: d 200000

```
$00200000 - 01010101 01010101 01010101 01010101 .................
$00200010 - 01010101 01010101 01010101 01010101 .................
$00200020 - 01010101 01010101 01010101 01010101 .................
$00200030 - 01010101 01010101 01010101 01010101 .................
$00200040 - 01010101 01010101 01010101 01010101 .................
$00200050 - 01010101 01010101 01010101 01010101 .................
$00200060 - 01010101 01010101 01010101 01010101 .................
```

39
Linking to a Module

The \( \text{l} \) command is used to link to a memory module. The address of the module is placed in relocation register 7. Subsequent \( \text{l} \) commands unlink the previously linked module and link to the new module.

The \( \text{l} \) and \( \text{a} \) commands (link and attach) work only when the system is up. To attach a module before the system comes up, use the \( \text{am} \) command.

### Table 2-12. \( \text{l} \) Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{l} ) &lt;module&gt;</td>
<td>Links debugger to the module specified by &lt;module&gt;</td>
</tr>
</tbody>
</table>

### Examples

RomBug: \( \text{l} \) term
RomBug: \( @7 \)
RomBug: \( .r \)

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000+r7</td>
<td>4AFC0001</td>
<td>display memory</td>
</tr>
<tr>
<td>0x00000010+r7</td>
<td>80000004</td>
<td></td>
</tr>
<tr>
<td>0x00000200+r7</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000300+r7</td>
<td>1B030123</td>
<td></td>
</tr>
<tr>
<td>0x00000400+r7</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000500+r7</td>
<td>18081808</td>
<td></td>
</tr>
<tr>
<td>0x00000600+r7</td>
<td>11110904</td>
<td></td>
</tr>
<tr>
<td>0x00000700+r7</td>
<td>7465726D</td>
<td></td>
</tr>
</tbody>
</table>

- \( J \) |
- \( z \) |
- \( p \) |
- \( E \) |
- \( h \) |
- \( . \) |
- \( d \) |
- \( scf.sc68681. \)
Symbolic Debugging

The debugger maintains a table of symbol modules containing an entry for each code module being debugged.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s</strong></td>
</tr>
<tr>
<td>Displays all symbols in all symbol modules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s [mod:]&lt;symb&gt;</strong></td>
</tr>
<tr>
<td>Displays a single symbol from the current symbol module or symbol module [mod:]. The * and ? wildcard symbols may be used in the symbol name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s[.]&lt;wcard&gt;</strong></td>
</tr>
<tr>
<td>Show symbols matching a wildcard expression in paged (default) or not paged [.] term mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sm</strong></td>
</tr>
<tr>
<td>Displays symbol module table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ss</strong></td>
</tr>
<tr>
<td>Sets current symbol module to the module containing the current PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ss &lt;addr&gt;</strong></td>
</tr>
<tr>
<td>Sets current symbol module to the module containing &lt;addr&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ss &lt;name&gt;:</strong></td>
</tr>
<tr>
<td>Sets current symbol module to the symbol module specified by &lt;name&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sd &lt;name&gt;</strong></td>
</tr>
<tr>
<td>Displays data symbols only for the specified symbol module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sc &lt;name&gt;</strong></td>
</tr>
<tr>
<td>Displays code symbols only for the specified symbol module</td>
</tr>
</tbody>
</table>

A symbol can be given as the parameter to the **s** command. Each symbol module is searched for the symbol and displayed if present. Note that symbols in symbol modules other than the current symbol module are prefixed by the name of the containing module.

The **ss** command with no parameter sets the current symbol module to the module containing the current program counter address:

RomBug: **ss**

default symbols belong to ‘progx’

The **s** command with no parameter displays all symbols in all symbol modules in the following format: the symbol name, a type code (D = data symbol, C = code symbol)
Using RomBug

and the absolute address of the symbol.

The asterisk (*) before the symbol module name indicates the current symbol module. Symbols without a symbol module qualifier are assumed to be in this symbol module. Symbols in other symbol modules are accessed by preceding the symbol name with a colon-terminated symbol module name.

OS-9 for 68K Example

<table>
<thead>
<tr>
<th>RomBug:</th>
<th>a ram</th>
<th>attach to RAM disk driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>RomBug:</td>
<td>s</td>
<td>display all symbols</td>
</tr>
<tr>
<td>ram:btext</td>
<td>C 0000B8C8 ram:Init</td>
<td>C 0000B938</td>
</tr>
<tr>
<td>ram:end</td>
<td>D 00010090 ram:Move</td>
<td>C 0000BA7C</td>
</tr>
<tr>
<td>ram:ReadSect</td>
<td>C 0000BA90 ram:WritSect</td>
<td>C 0000BA9C</td>
</tr>
<tr>
<td>ram:Seek</td>
<td>C 0000BAAC ram:etext</td>
<td>C 0000BABC</td>
</tr>
<tr>
<td>ram:PutStat</td>
<td>C 0000BAC8 ram:GetStat</td>
<td>C 0000BAC8</td>
</tr>
<tr>
<td>ram:Termt</td>
<td>C 0000BAD2 ram:bname</td>
<td>C 0000BACE</td>
</tr>
<tr>
<td>RomBug:</td>
<td>sm</td>
<td>show symbol module table</td>
</tr>
<tr>
<td>Mod Addr</td>
<td>Code Lo</td>
<td>Code Hi</td>
</tr>
<tr>
<td>001f0700</td>
<td>0000B8C8</td>
<td>0000BAF2</td>
</tr>
<tr>
<td>RomBug:</td>
<td>b ReadSect</td>
<td>set breakpoint at ReadSect</td>
</tr>
<tr>
<td>RomBug:</td>
<td>sm</td>
<td>show modules (current (*) is ram)</td>
</tr>
<tr>
<td>Mod Addr</td>
<td>Code Lo</td>
<td>Code Hi</td>
</tr>
<tr>
<td>001f0700</td>
<td>0000B8C8</td>
<td>0000BAF2</td>
</tr>
<tr>
<td>RomBug:</td>
<td>a kernel</td>
<td>attach to kernel module</td>
</tr>
<tr>
<td>RomBug:</td>
<td>s</td>
<td>show all symbols</td>
</tr>
<tr>
<td>btext</td>
<td>C 0000B8C8 Init</td>
<td>C 0000B938</td>
</tr>
<tr>
<td>end</td>
<td>D 00010090 Move</td>
<td>C 0000BA7C</td>
</tr>
<tr>
<td>ReadSect</td>
<td>C 0000BA90 WritSect</td>
<td>C 0000BA9C</td>
</tr>
<tr>
<td>Seek</td>
<td>C 0000BAAC etext</td>
<td>C 0000BABC</td>
</tr>
<tr>
<td>PutStat</td>
<td>C 0000BAC8 GetStat</td>
<td>C 0000BAC8</td>
</tr>
<tr>
<td>Termt</td>
<td>C 0000BAD2 bname</td>
<td>C 0000BACE</td>
</tr>
<tr>
<td>kernel:end</td>
<td>D 0000FPPP kernel:btext</td>
<td>C 00003000</td>
</tr>
<tr>
<td>kernel:_syscmmt</td>
<td>D 0000FPPP</td>
<td>kernel:CopyRight</td>
</tr>
<tr>
<td>kernel:MFUtype</td>
<td>C 0000307A</td>
<td>kernel:SysErrMsg</td>
</tr>
<tr>
<td>kernel:PCMMsg</td>
<td>C 00003193</td>
<td>kernel:ResetMsg</td>
</tr>
<tr>
<td>kernel:EOL</td>
<td>C 000031B4 kernel:Cold</td>
<td>C 000031B8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel:Wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel:etext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel:RetProc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel:bname</td>
</tr>
<tr>
<td>RomBug:</td>
<td>sm</td>
<td>show symbol modules</td>
</tr>
<tr>
<td>Mod Addr</td>
<td>Code Lo</td>
<td>Code Hi</td>
</tr>
<tr>
<td>00006cb4</td>
<td>00003000</td>
<td>00006cb2</td>
</tr>
<tr>
<td>001f0700</td>
<td>0000B8C8</td>
<td>0000baf2</td>
</tr>
<tr>
<td>RomBug:</td>
<td>s ReadSect</td>
<td>find a particular symbol</td>
</tr>
<tr>
<td>ReadSect</td>
<td>C 0000BA90</td>
<td></td>
</tr>
<tr>
<td>RomBug:</td>
<td>s kernel:Cold</td>
<td>find a symbol qualified with module name</td>
</tr>
<tr>
<td>kernel:Cold</td>
<td>C 000031B8</td>
<td></td>
</tr>
<tr>
<td>RomBug:</td>
<td>s <em>R</em></td>
<td>find symbol using wildcard</td>
</tr>
<tr>
<td>0000BA90</td>
<td>kernel:ResetMsg</td>
<td>C 0000319F</td>
</tr>
<tr>
<td>kernel:RtnPd</td>
<td>C 000060BC</td>
<td>kernel:R64</td>
</tr>
<tr>
<td>kernel:RetTime</td>
<td>C 00006754</td>
<td>kernel:RemChild</td>
</tr>
<tr>
<td>kernel:RetProc</td>
<td>C 00006C96</td>
<td></td>
</tr>
<tr>
<td>RomBug:</td>
<td>ss</td>
<td>set default symbols to 'RAM'</td>
</tr>
<tr>
<td>default symbols belong to 'ram'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# OS-9 Example

RomBug: `s

<table>
<thead>
<tr>
<th>Mod Addr</th>
<th>Code Lo</th>
<th>Code Hi</th>
<th>Data Lo</th>
<th>Data Hi</th>
<th>Count</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>003d399a</td>
<td>003c0000</td>
<td>003d298e</td>
<td>00010000</td>
<td>0001a200</td>
<td>544</td>
<td>boot</td>
</tr>
</tbody>
</table>

RomBug: `sm

<table>
<thead>
<tr>
<th>Mod Addr</th>
<th>Code Lo</th>
<th>Code Hi</th>
<th>Data Lo</th>
<th>Data Hi</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>003d399a</td>
<td>003c0000</td>
<td>003d298e</td>
<td>00010000</td>
<td>0001a200</td>
<td>boot</td>
</tr>
</tbody>
</table>

RomBug: `s

<table>
<thead>
<tr>
<th>Mod Addr</th>
<th>Code Lo</th>
<th>Code Hi</th>
<th>Data Lo</th>
<th>Data Hi</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>003c612a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RomBug: `s boot:main

<table>
<thead>
<tr>
<th>Mod Addr</th>
<th>Code Lo</th>
<th>Code Hi</th>
<th>Data Lo</th>
<th>Data Hi</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>003c612a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Attaching a Module

The `a` command associates a symbol module with the given code module. The code module data and code addresses can then be referenced symbolically. The syntax for the `a` command is shown in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code> [&lt;module&gt;]</td>
<td>Attaches to the specified module</td>
</tr>
<tr>
<td><code>am</code> &lt;beg&gt; &lt;end&gt;</td>
<td>Attaches all modules found in the address range specified by &lt;beg&gt; to &lt;end&gt;</td>
</tr>
</tbody>
</table>

RomBug attempts to attach (link) the `STB` symbol module for each module given as arguments. The module must already exist in memory. If the symbol module is not in memory, an error is returned. If the `am` command is given, RomBug searches for any symbolically and associated code modules found in the specified address range.

The `l` and `a` commands (link and attach) work only when the system is up. To attach a module before the system comes up, use the `am` command.
Example
RomBug: a
RomBug: a rb1772
RomBug: a rbf rb1772
RomBug: am 70000 90000

Viewing Expressions
The v command evaluates and displays any legal expression in decimal, hexadecimal, and as a symbolic address.

Table 2-15. v Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v &lt;expr&gt;</td>
<td>Evaluates an expression and prints the value in decimal, hexadecimal, and as a symbolic address</td>
</tr>
</tbody>
</table>

Example 1

```
dn: 0000000C 000C0064 00000080 00000003 00000000 000001050 00000000
an: 00000000 00015F80 00000000 000F32A0 00000000 00015F30 00015F30
pc: 000F32E6 cc: 00 -------
<68881 in Null state>
_cstart             >2D468010         move.l d6,_totmem(a6)
RomBug: v .d1
0x000C0064 (786532) 0xC0064
RomBug: v .pc
0x000F32E6 (996078) _cstart
RomBug: v .pc+10
0x000F32FE (996094) _cstart+0x10
RomBug: v .pc+400
0x000F36E6 (997102) _initarg+0x38
RomBug: v .d1
0x000C0064 (786532) 0xC0064
RomBug: v .d5
0x000000A2 (162) 0xA2
RomBug: v .d5+4
0x0000000A (10) 0xA
RomBug: v .sp+8
0x00015EE4 (89828) 0x15EE4
RomBug: gs main
Installed symbol module for trap handler ‘cio’
dn: 00000001 00015F4E 00015F4A 00015F56 00000000 000000A2 00001050 00000000
an: 000152DE 001F214A 00015F4E 00015F4A 00015F46 00015F42 00000000 0001CF30 00015FED4
pc: 000F34F0 cc: 00 -------
<68881 in Null state>
main               >4BE7F080         movem.l d0-d3/a0,-(a7)
RomBug: v .a1
0x001F214A (2040138) cio:CIOTrap
RomBug: v .d1
0x00015F4B (89934) 0x15F4B
RomBug: v [.d1]
0x00015F46 (89926) 0x15F46
RomBug: dl [.d1]
0x15F46
```

Table 2-15. v Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v &lt;expr&gt;</td>
<td>Evaluates an expression and prints the value in decimal, hexadecimal, and as a symbolic address</td>
</tr>
</tbody>
</table>
Example 2

dn:00000000 0012345 00000000 0000004A 00000000 0000FA08 00000000 00000000
an:00008164 00000000 00000045 00000000 00000000 00000000 00000000 00002044 00000000
pc: 00000000 cc: 00  (-----)
<6888 in Null state>
dis: v .dl
0x00012345 (74565) 0x00012345
RomBug: v .d5>3
0x000001F41 (8001) 0x000001F41
RomBug: v li+69
0x0000007A (122) 0x0000007A
RomBug: v #li+69
0x00000050 (80) 0x00000050
RomBug: v fe61*2
Symbol 'fe61*2' not found
RomBug: v 0xfe61*2
0x0001FCC2 (130242) 0x0001FCC2
RomBug: v .a0-24
0x000008140 (33088) 0x000008140
RomBug: v .a0&fff
0x00000164 (356) 0x00000164
RomBug: v .d5-5*.a2
0x004362CF (4416207) 0x004362CF
RomBug: v .d5-(5*.a2)
0x0000FB8F (63663) 0x0000FB8F
RomBug: v .a2^..d3
0x0000000F (15) 0x0000000F
RomBug: d1 .a6
memory indirection
0x00002044 - 001FEP70 001FF4F4 00156440 000023E4 ..op..tt..d@..#d
0x001FEP70 [092912] 0x001FEP70
RomBug: v [.a6]+10
0x001FEP80 (2092928) 0x001FEP80
RomBug: v [.a6]w
0x0000001F (31) 0x0000001F
RomBug: v [.a6]w+1
0x00000020 (32) 0x00000020
RomBug: v [.a6]b
0x00000000 (0) 0x00000000 Command

OEMCMD

Oemcmd is a system module that can extend the functionality of RomBug for non-68K systems. To enable oemcmd, it must be loaded into memory and either added to your customization module list in your init module, or you can use the p2init utility to initialize it. Once initialized, you will have access to the extra functionality in RomBug.

Table 2-16. x Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xc [pid]</td>
<td>Display current [or pid] process information</td>
</tr>
<tr>
<td>xf</td>
<td>Display free memory information (similar to mfree -e)</td>
</tr>
<tr>
<td>xm</td>
<td>Display root module directory (similar to mdir -e)</td>
</tr>
</tbody>
</table>
Using RomBug

Examples

xc can show information on the current running process when RomBug was called.

RomBug: xc
Descriptor address: 0xc8df7910
Module Name : break
Process ID : 6
Parent Process ID : 2
Waiting thread ID : 0
Process priority : 128
Process age : 128
Process status / queue : 0x00000210 / *
Process exit status : 0x00000000
System Stack Pointer : 0xc8dfba48
System Stack Base : 0xc8dfba90
User Stack Pointer : 0xc72a73a8
System State Exception Recovery PC : 0xc702c864
System State Exception Recovery Stack : 0xc8dfba3c
Signal flag : 000, level : 000, count = 000, last = 000

By giving xc an option you can see information on a particular process.

RomBug: xc 2
Descriptor address: 0xc8e68c20
Module Name : mshell
Process ID : 2
Parent Process ID : 0
Waiting thread ID : 0
Process priority : 128
Process age : 128
Process status / queue : 0x00000200 / w
Process exit status : 0x00000000
System Stack Pointer : 0xc8e6cda0
System Stack Base : 0xc8e6cda0
User Stack Pointer : 0xc8e6cda0
System State Exception Recovery PC : 0xc702c864
System State Exception Recovery Stack : 0xc8e6cda3c
Signal flag : 000, level : 000, count = 000, last = 000

xf will display information similar to mfree -e.

RomBug: xf
Free memory map:

Table 2-16. x Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xp[a]</td>
<td>Display processes, a = display alternate data (similar to procs -e[a])</td>
</tr>
<tr>
<td>xq</td>
<td>Display process queue information</td>
</tr>
<tr>
<td>xs</td>
<td>Display OS-9 system globals</td>
</tr>
<tr>
<td>xw &lt;addr&gt;</td>
<td>Display module that contains specified address</td>
</tr>
<tr>
<td>x?</td>
<td>Display this help</td>
</tr>
</tbody>
</table>

Table 2-16. x Command
Chapter 2: Commands

<table>
<thead>
<tr>
<th>Segment Address</th>
<th>Size of Segment</th>
<th>Color</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c72c0000</td>
<td>$01b08000</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>$c84ec000</td>
<td>$00004000</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>$c8df4000</td>
<td>$00003000</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>$c8e00000</td>
<td>$00001000</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>$c8f00000</td>
<td>$000e0000</td>
<td>128</td>
<td>0</td>
</tr>
</tbody>
</table>

Number of memory segments: 5
Current total free RAM: 29294592 bytes

_xq_ will display processes that are in the active, sleep, or wait queues.

RomBug: _xq_
Process queue header information

active queue:

sleep queue:
3 inetd
5 spfndpd

wait queue:
2 mshell

_xs_ displays OS-9 system globals with a description.

RomBug: _xs_
OS-9 system globals: 0xc729ad70

- _d_id_ = 0xff6a - sync code (system globals ID)
- _d_mputyp_ = 0x00000004 - MPU type
- _d_fputyp_ = 0x00000000 - FPU type
- _d_compat_ = 0x0040 - compatibility/control flags
- _d_minpty_ = 0 - system minimum priority
- _d_maxage_ = 0 - system maximum natural age
- _d_maxsigs_ = 32 - default maximum numbers of signals queued
- _d_totram_ = 30920960 - total RAM available at startup
- _d_blksiz_ = 4096 - system minimum allocatable block size
- _d_minblk_ = 16 - process minimum allocatable block size
- _d_tick_ = 80 - current tick (count down tick)
- _d_tcksec_ = 100 - clock tickrate (number of ticks per second)
- _d_slice_ = 2 - current time slice remaining
- _d_tslice_ = 2 - ticks per slice
- _d_time_ = 0xbf26a46 - system time: seconds since reference date
- _d_ticks_ = 0x00002468 - system heartbeat (current tick counter)
- _d_d_unkirq_ = 0 - unknown IRQ count (unserviced IRQ count)
- _d_init_ = 0xc7041a20 - pointer to initialization module
- _d_system_ = 0xc7008180 - Bootstrap ROM information structure pointer
- _d_mdroot_ = 0xc72a2e74 - system module directory root node pointer
- _d_shmdroot_ = 0x00000000 - shared module directory root node pointer
- _d_prdctbl_ = 0xc8eaf2f0 - process descriptor block table pointer
- _d_proc_ = 0xc87df910 - pointer to current process descriptor

_xw_ will display the name of a module contained at a particular address, and the offset into the module. If no module is located at that address, a message will be printed stating no module contains the address <addr>.

RomBug: _xw_ c7130000
C712e760+0x000018a0 setime
RomBug: _xw_ 00800000
No module contains the address 0x00800000
This chapter includes information on 68xxx processor-specific information.

- **-o Option**
- **Commands**
- **Supported Registers**
- **Display Information**
- **Change Machine Registers**
- **Instruction Disassembly Memory Display**
- **Floating Point Memory Display**
- **Setting and Displaying Debug Options**
-o Option

The -o options defined in the following table are supported by the 68xxx version of RomBug.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Toggle control registers</td>
</tr>
<tr>
<td>c&lt;n&gt;[:f]</td>
<td>Set MPU type to &lt;n&gt; where &lt;n&gt; = processor number and FPCP to 6888&lt;f&gt;</td>
</tr>
<tr>
<td>d</td>
<td>Toggle FPCP decimal register display</td>
</tr>
<tr>
<td>e&lt;addr&gt;</td>
<td>Display exception frame (default &lt;addr&gt; is .a7)</td>
</tr>
<tr>
<td>f</td>
<td>Toggle FPCP register display</td>
</tr>
<tr>
<td>m</td>
<td>Toggle MMU register display</td>
</tr>
</tbody>
</table>

Commands

68xxx-specific command information is provided in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mf[s][n] &lt;beg&gt; &lt;end&gt; &lt;value&gt;</td>
<td>[n] indicates a non-aligned fill. N is required for the 68020 and 68030 processors for non-word aligned.</td>
</tr>
<tr>
<td>e</td>
<td>Enable/disable monitoring of processor-specific default exception vectors. Default vector numbers for exceptions are: Bus error = 2 Address error = 3 Illegal instruction = 4</td>
</tr>
</tbody>
</table>

Supported Registers

The following tables define registers supported in general and by specific processors. Processor registers can be changed with the dot (.) command. Any processor register or coprocessor control register can be changed with this command:

}.${regname} <expr>
where `<regname>` is any of the register names in the following register tables.

Expressions and the `v` command use the register names listed in the following register tables to obtain the value of a register.

<table>
<thead>
<tr>
<th>Table 3-3. 68xxx Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register</strong></td>
</tr>
<tr>
<td>.d0 - .d7</td>
</tr>
<tr>
<td>.a0 - .a7</td>
</tr>
<tr>
<td>.usp</td>
</tr>
<tr>
<td>.ssp</td>
</tr>
<tr>
<td>.sp</td>
</tr>
<tr>
<td>.cc</td>
</tr>
<tr>
<td>.sr</td>
</tr>
<tr>
<td>.pc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3-4. 68010/20/30/40/60 Only Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register</strong></td>
</tr>
<tr>
<td>.vbr</td>
</tr>
<tr>
<td>.sfc</td>
</tr>
<tr>
<td>.dfc</td>
</tr>
<tr>
<td>.cacr</td>
</tr>
</tbody>
</table>
### Table 3-5. 68020/30/40 Only Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.msp</td>
<td>Master stack pointer</td>
</tr>
<tr>
<td>.isp</td>
<td>Interrupt stack pointer</td>
</tr>
<tr>
<td>.caar</td>
<td>Cache address register</td>
</tr>
</tbody>
</table>

### Table 3-6. 68030/68551 Only Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.crp</td>
<td>CPU root pointer</td>
</tr>
<tr>
<td>.srp</td>
<td>Supervisor root pointer</td>
</tr>
<tr>
<td>.tc</td>
<td>Translation control register</td>
</tr>
</tbody>
</table>

### Table 3-7. 68020/30 with 68881/82 Only or 68040/60 Only Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.fp0 -.fp7</td>
<td>Floating point data registers</td>
</tr>
<tr>
<td>.fpsr</td>
<td>Floating point status register</td>
</tr>
<tr>
<td>.fpcr</td>
<td>Floating point control register</td>
</tr>
<tr>
<td>.fpiar</td>
<td>Floating point instruction address register</td>
</tr>
</tbody>
</table>
### Table 3-8. 68030 Only Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.tt0</td>
<td>Transparent translation register</td>
</tr>
<tr>
<td>.tt1</td>
<td>Transparent translation register</td>
</tr>
<tr>
<td>mmusr</td>
<td>MMU status register</td>
</tr>
</tbody>
</table>

### Table 3-9. 68040/60 Only Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.dtt1</td>
<td>Data transparent translation register</td>
</tr>
<tr>
<td>.dtt0</td>
<td>Data transparent translation register</td>
</tr>
<tr>
<td>.itt1</td>
<td>Instruction transparent translation register</td>
</tr>
<tr>
<td>.itt0</td>
<td>Instruction transparent translation register</td>
</tr>
<tr>
<td>.urp</td>
<td>User root pointer register</td>
</tr>
<tr>
<td>.srp</td>
<td>Supervisor root pointer register</td>
</tr>
<tr>
<td>.tcr</td>
<td>Translation control register</td>
</tr>
</tbody>
</table>

### Table 3-10. 68060 Only Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.pcr</td>
<td>Processor configuration register</td>
</tr>
<tr>
<td>.buscr</td>
<td>Bus control register</td>
</tr>
</tbody>
</table>
Display Information

Register information shown below displays following an \texttt{oa} command.

\begin{verbatim}
vbr:00000000 sfc:7=CS dfc:7=CS caar:1=.E caar:80000000
dn:00000000 001DE3A 00000001 00000000 00000000 01010101 000000D6 01010101
an:0010E9C9 0001B6B4 00000000 00000000 001F31A0 00000000 00022DA0 0001CBEE
pc:0010E3DC sr:2000 (--S--0-----)t:OFF msp:C001A4FC usp:0001DD98 ^isp^<68881 in Null state>
\end{verbatim}

If the host processor type is a 68010/20/30/40/60/CPU32, the first line is the control register display. For the 68060 processors, a second line displays the \texttt{pcr} and \texttt{buscr} registers. The \texttt{oa} option toggles the display of this line. The first three registers are displayed for the 68010/20/30/40/60/CPU32. \texttt{vbr} is the vector base register. \texttt{sfc} and \texttt{dfc} are the source and destination function code registers, respectively. The three bit value of the \texttt{sfc}/\texttt{dfc} registers is interpreted as identified in the following table.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Value} & \textbf{Text} & \textbf{Meaning} \\
\hline
0 & ?? & Undefined \\
1 & UD & User data space \\
2 & UP & User program space \\
3 & ?? & Undefined \\
4 & ?? & Undefined \\
5 & SD & Supervisor data space \\
\hline
\end{tabular}
\caption{\texttt{sfc}/\texttt{dfc} Register Three Bit Value}
\end{table}
Table 3-12. sfc/dfc Register Three Bit Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Text</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>SP</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>7</td>
<td>CS</td>
<td>CPU space</td>
</tr>
</tbody>
</table>
If the host processor is a 68020, 68030, 68040, or 68060, the cache control register (cacr) and cache address register (caar) are displayed. The cacr is interpreted as follows:

**68020:**

<table>
<thead>
<tr>
<th>cacr:0</th>
<th>E (or .) = Cache enabled (disabled)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F (or .) = Cache frozen (unfrozen)</td>
</tr>
</tbody>
</table>

Value of cacr

**68030:**

<table>
<thead>
<tr>
<th>cacr:3011</th>
<th>Data Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>E (or .)</td>
<td>Cache enabled (disabled)</td>
</tr>
<tr>
<td>F (or .)</td>
<td>Cache frozen (unfrozen)</td>
</tr>
<tr>
<td>B (or .)</td>
<td>Burst fill enabled (disabled)</td>
</tr>
<tr>
<td>W (or .)</td>
<td>Write allocate mode on (off)</td>
</tr>
</tbody>
</table>

Value of cacr

**68040:**

<table>
<thead>
<tr>
<th>cacr:80000000</th>
<th>(DE/ID)</th>
</tr>
</thead>
</table>

Instruction cache enabled (disabled)
Data cache enabled (disabled)

**68060:**

<table>
<thead>
<tr>
<th>cacr:A0808000</th>
<th>(E.S...E...E...)</th>
</tr>
</thead>
</table>

Value of cacr

- H = 1/2 cache mode (full)
- A = No allocate mode (allocate)
- E = Enable instruction cache (disable)
- U = Clear user branch cache
- C = Clear all branch cache
- E = Enable branch cache (disable)
- S = Enable store buffer (disable)
- A = No allocate mode (allocate)
- E = Enable data cache (disable)

For the 68349, the CIC control registers are not displayed by the `oa` command. Use the memory examine commands to examine the CIC MCR Register.
For the 68060 processor, the `pcr` and `buscr` are interpreted as follows:

$$pcr = 04300101 \text{ (68060:Rev-1:DDBG:EFP:ESS)}$$

- Bits 31 - 16 define the processor identification
- Bits 15 - 8 define the processor revision
- Bits 7 - 0 are as follows:

```
+----------------+-------------------+-------------------+
|    Reserved    | ESS enable superscalar operation (disable) |
|                | DFP disable floating point unit (enable)   |
|                | EDBG enable debug functions (disable)      |
```

$$buscr: 0000 0000 \text{ (. . . . . . .)}$$

```
+----------------+-------------------+-------------------+
| Reserved       | SLE = Shadow copy, lock end |
|                | LE = Lock end       |
|                | SL = Shadow copy, lock |
|                | L = Lock            |
```

The `dn` and `an` fields display the data and address registers from `d0-d7` and `a0-a7` respectively. The last line displays the program counter (`pc`), status register (`sr`), and stack pointers (`msp`, `usp`, and `isp`).

The status register value is interpreted as follows:

$$sr:0000 \text{ (-----0-----) t:OFF}$$

```
+----------------+-------------------+-------------------+
| C = Carry bit set | V = Overflow bit set |
| Z = Zero bit set  | N = Negative bit set |
| X = Extend bit set| 0-7 = Interrupt priority mask |
| ? = Reserved      | M = Master, I = Interrupt state (68020/30/40 only) |
|                  | S = Supervisor, U = User state |
|                  | T = Flow change trace (68020/30/40 and CPU32 only) |
|                  | T = Trace |
| t:OFF - No tracing | t:CHG - Trace on change of flow |
| t:ALL - Trace all instructions |
| t:??? - Undefined (T0 and T1 both set) |
```
The current stack pointer is always displayed in the A7 register position. The ^sp^ indicator specifies the current stack pointer in use. The stack pointers not currently in use are displayed in alternate field positions. The 68020/30/40 have three stack pointer registers. The 68000/10/60/70 and CPU32 have only two stack pointer registers.

Following are examples of stack pointer register displays.

**68000, 68010, 68060, 68070, and CPU32 in user state:**

```
an:0010E9C9 0001B6B4 00000000 00000000 001F31A0 00000000 00022DA0 0001:
```
```
   pc:0010E3DC  sr:2000 (--U-0-----)                   ssp:0001CBEE   ^u
```

**68000, 68010, 68060, 68070, and CPU32 in supervisor state:**

```
an:0010E9C9 0001B6B4 00000000 00000000 001F31A0 00000000 00022DA0 0001
```
```
   pc:0010E3DC  sr:2000 (--S--0-----)                    usp:0001DD98   ^s
```

**68020, 68030, and 68040 in user state:**

```
an:0010E9C9 0001B6B4 00000000 00000000 001F31A0 00000000 00022DA0 0001I
```
```
   pc:0010E3DC  sr:2000 (--UI-0-----)t:OFF isp:001CBEE msp:C001A4FC   ^u
```

**68020, 68030, and 68040 in supervisor state (interrupt stack):**

```
an:0010E9C9 0001B6B4 00000000 00000000 001F31A0 00000000 00022DA0 0001
```
```
   pc 0010E3DC sr:2000 (--SI-0-----)t:OFF msp:C001A4FC usp:001DD98   ^s
```

**68020, 68030, and 68040 in supervisor state (master stack):**

```
an:0010E9C9 0001B6B4 00000000 00000000 001F31A0 00000000 00022DA0 C001:
```
```
   pc:0010E3DC  sr:3000 (--SM-0-----)t:OFF usp:001DD98 isp:001CBEE    ^ms
```

The following discussion of the 68881/82 coprocessor and 68040 FPU registers applies only to OS-9 for 68K systems running on a 68020/30/40/60 processor with a floating point unit. The examples shown use the 68881 coprocessor.

If the display floating point registers option is set, the next line(s) indicate the state of the floating point unit. If the FPU is not present on the system, the following message displays:

<No FPCP available>

If the FPU is in its initial reset state, the following message displays:

<FPCP in Null state>
When the FPU is accessed, a floating point register dump displays. If the setting of the debugger decimal register display option indicates hex display, the FPU registers display:

```
f0: 40010000 D555555S 55552000 fp4: 7FFF0000 FFFFFFFF FFFFFFFF fpcr: 00
fp1: 7FFF0000 FFFFFFFF FFFFFFFF fp5: 7FFF0000 FFFFFFFF FFFFFFFF fpiar: 0
fp2: 7FFF0000 FFFFFFFF FFFFFFFF fp6: 7FFF0000 FFFFFFFF FFFFFFFF fpsr: 00
fp3: 7FFF0000 FFFFFFFF FFFFFFFF fp7: 7FFF0000 FFFFFFFF FFFFFFFF (----
```

If a decimal display is indicated, the registers display in the following format:

```
fp0: 6.666666666666666 fp4: <NaN> fpcr: 000
fp1: <NaN> fp5: <NaN> fpiar: 000
fp2: <NaN> fp6: <NaN> fpsr: 000
fp3: <NaN> fp7: <NaN> (----
```

The value of the registers is printed in decimal using scientific notation when the value becomes very large or very small. IEEE not-a-number values are printed as <NaN>, plus and minus infinity values are printed as <+Inf> and <-Inf>, respectively. The extended precision values are converted to double precision before printing, potentially resulting in conversion overflow. The hexadecimal format display can be used to determine the exact values in the registers.

The eight floating point registers are displayed in either hex or decimal form depending on the floating point register display option setting.

The floating point status registers display to the far right of the display:

```
fpcr: 0000 -- floating point control register
fpiar: 00000000 floating point instruction address register
fpsr: 00000000 floating point status register
(---- 0) fpsr interpretation bits
```

The -- field next to the fpcr register displays an interpretation of the FPU rounding mode and precision. These fields are interpreted as follows:

```
 fpcr: 0000 --

Rounding mode:
N = Nearest
Z = Toward zero
- = Toward minus infinity
+ = Toward plus infinity

Rounding Precision:
X = Extended
S = Single
D = Double
? = Undefined
```
The \textit{fpsr} condition code and quotient bytes are displayed as follows:

\begin{itemize}
  \item Quotient byte value (displays in signed decimal)
  \item \( ? \) = NaN or unordered
  \item \( I \) = Infinity
  \item \( Z \) = Zero
  \item \( M \) = Negative
\end{itemize}

Immediately following the main floating register display, the debugger interprets the exception enable byte of the control register and the exception status and accrued exception bytes of the status register. If all bits in the byte are zero, nothing is printed. Otherwise the bits are displayed as follows:

\begin{itemize}
  \item \textbf{XS:} (BSUN, SNAN, OPERR, OVFL, UNFL, DZ, INEX2, INEX1) \textit{fpsr exception status}
  \item \textbf{AX:} (IOP, OVFL, UNFL, DZ, INEX, ???, ???, ???) \textit{fpsr accrued exception}
  \item \textbf{XE:} (BSUN, SNAN, OPERR, OVFL, UNFL, DZ, INEX1) \textit{fpcr exception enable}
\end{itemize}

A full register display example follows:

\begin{verbatim}
  dn:00000000 00000000 00000001 00000000 00000000 000000A2 00000105 00000000
  an:000152D2 00000000 00015F4E 00015F46 00015F42 00000000 0001CF30 0001ED4
  pc:00139364  cc: 04 (--Z--)  
  fp0:40010000 C0000000 00000000 fp1:7FF0000 FFFFFFFF FFFFFFFF  fpcr: 0000 XN
  fp2:7FF0000 FFFFFFFF FFFFFFFF fp5:7FF0000 FFFFFFFF FFFFFFFF fpiar: 00000000
  fp3:7FF0000 FFFFFFFF FFFFFFFF fp6:7FF0000 FFFFFFFF FFFFFFFF fpsr: 00000008
  fp4:7FF0000 FFFFFFFF FFFFFFFF fp7:7FF0000 FFFFFFFF FFFFFFFF (---- 0)
  AX:(INEX)
  _exit+0x6           >DEADDEAD         add.l -8531(a5),d7
\end{verbatim}

When using the FPU in system state, note that the operating system preserves the state of the FPU for user processes only. If system-state code wishes to access the FPU, its full context must be preserved before and after use.
The actual stack pointer accessed given the register name and processor state is shown in the following table.

Table 3-13. Stack Pointer Names by State/Processor

<table>
<thead>
<tr>
<th>Register</th>
<th>All / User</th>
<th>68000/10/60 /70/CPU32 Supervisor</th>
<th>68020/30/40 Supervisor-IRQ</th>
<th>68020/30/40 Supervisor-Master</th>
</tr>
</thead>
<tbody>
<tr>
<td>.a7</td>
<td>a7/usp</td>
<td>a7/ssp</td>
<td>isp</td>
<td>msp</td>
</tr>
<tr>
<td>.sp</td>
<td>a7/usp</td>
<td>a7/ssp</td>
<td>isp</td>
<td>msp</td>
</tr>
<tr>
<td>.ssp¹</td>
<td>ssp</td>
<td>a7/ssp</td>
<td>isp</td>
<td>msp</td>
</tr>
<tr>
<td>.usp</td>
<td>a7/usp</td>
<td>a7/usp</td>
<td>usp</td>
<td>usp</td>
</tr>
<tr>
<td>.isp²</td>
<td>isp</td>
<td>-</td>
<td>isp</td>
<td>isp</td>
</tr>
<tr>
<td>.msp²</td>
<td>msp</td>
<td>-</td>
<td>msp</td>
<td>msp</td>
</tr>
</tbody>
</table>

¹Current supervisor stack pointer - all states
²User state is not applicable on 68000/10

The MMU registers are a special case; these registers can be displayed but not changed. The om command toggles the following display:
The floating point register change command allows the change value to be either a double precision decimal constant or a left-justified hexadecimal value:

```
.fp<n> <float-decimal constant>
```

or

```
.fp<n> <96-bit left-justified hex constant>
```

or

```
.fp<n> .fp<n>
```

<n> is one of 0 - 7 representing the desired general floating point register.

The syntax for `<float-decimal constant>` is:

```
[±]digits[.digits][Ee[±]integer]
```

The syntax for `<96-bit left-justified hex constant>` is:
\[0xh\]

*h* represents up to 12 hexadecimal digits. If less than 12 digits are given, the value is padded on the right with zeroes.

Bits 68-80 of an extended precision value in IEEE are always zero.

**Examples**

```
0xh

dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 0025610 0001E770
pc:000F8FA8 cc: 04 (--Z--)
_exit+0x6 >DEADDEAD add.l -8531(a5),d7
RomBug: .d4 100
RomBug:.

```

set D4 to 100
display registers

```
0xh

dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 0025610 0001E770
pc:000F8FA8 cc: 04 (--Z--)
RomBug: .d4 .d2+.a6
RomBug:.

```

set D4 using an expression
display registers

```
0xh

dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 0025610 0001E770
pc:000F8FA8 cc: 04 (--Z--)
RomBug: .fp0 4 set FP0 to 4.0
RomBug:.

```

set FP0 with hex value (left justified)

```
0xh

dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 0025610 0001E770
pc:000F8FA8 cc: 04 (--Z--)
fp0:40010000 80000000 00000000 fp4:7FF00000 FFFFFF00 FFFFFFFF fpcr: 0000 XN
fp1:7FF00000 FFFFFF00 FFFFFFFF fp5:7FF00000 FFFFFFFF fpiar: 00000000
fp2:7FF00000 FFFFFFFF FFFFFFFF fp6:7FF00000 FFFFFF fpsr: 00000000
fp3:7FF00000 FFFFFFFF FFFFFFFF fp7:7FF00000 FFFFFF fpcr: 00000000 fpcr: 00000000
 AX:(INEX) XS:(INEX2)
_exit+0x6 >DEADDEAD add.l -8531(a5),d7
RomBug: .fp0 0x4
RomBug:.

```

set FP0 to 0x4

```
0xh

dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 0025610 0001E770
pc:000F8FA8 cc: 04 (--Z--)
fp0:40000000 80000000 00000000 fp4:7FF00000 FFFFFF00 FFFFFFFF fpcr: 0000 XN
fp1:7FF00000 FFFFFFFF FFFFFFFF fp5:7FF00000 FFFFFFFF fpiar: 00000000
fp2:7FF00000 FFFFFFFF FFFFFFFF fp6:7FF00000 FFFFFF fpsr: 00000000
fp3:7FF00000 FFFFFFFF FFFFFFFF fp7:7FF00000 FFFFFF fpcr: 00000000 fpcr: 00000000
 AX:(INEX) XS:(INEX2)
_exit+0x6 >DEADDEAD add.l -8531(a5),d7
Using RomBug

```plaintext
Using RomBug

- Change Machine Registers
- Examples

Change Machine Registers

- Bits 68-80 of an extended precision value in IEEE are always zero.

Examples

```plaintext
Examples

```
RomBug: .d4 .d2+.a6
RomBug: .
dn:00000000 00002700 00000002 00020000 00010002 00000001 FFFFE000 00004C00
an:00015100 FFFFE2800 00010000 00015100 0005C00 000150F8 00010000 000150B4
pc:003C51A4 sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00000000 ^isp^
fp0:<NaN>                     fp4:<NaN>                      fpcr: 0000 XN
fp1:<NaN>                     fp5:<NaN>                     fpiar: 00000000
fp2:<NaN>                     fp6:<NaN>                      fpsr: 00000000
fp3:<NaN>                     fp7:<NaN>                      (----    0)
null_restore        >F36E050C         frestore fp_cir+$2(a6)

RomBug: .fp0 4
RomBug: .
dn:00000000 00002700 00000002 00020000 00010002 00000001 FFFFE000 00004C00
an:00015100 FFFFE2800 00010000 00015100 0005C00 000150F8 00010000 000150B4
pc:003C51A4 sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00000000 ^isp^
fp0:4                         fp4:<NaN>                      fpcr: 0000 XN
fp1:<NaN>                     fp5:<NaN>                     fpiar: 00000000
fp2:<NaN>                     fp6:<NaN>                      fpsr: 00000000
fp3:<NaN>                     fp7:<NaN>                      (----    0)
null_restore        >F36E050C         frestore fp_cir+$2(a6)

RomBug: .fp0 3.14159
RomBug: .
dn:00000000 00002700 00000002 00020000 00010002 00000001 FFFFE000 00004C00
an:00015100 FFFFE2800 00010000 00015100 0005C00 000150F8 00010000 000150B4
pc:003C51A4 sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00000000 ^isp^
fp0:3.14159                   fp4:<NaN>                      fpcr: 0000 XN
fp1:<NaN>                     fp5:<NaN>                     fpiar: 00000000
fp2:<NaN>                     fp6:<NaN>                      fpsr: 00000000
fp3:<NaN>                     fp7:<NaN>                      (----    0)
null_restore        >F36E050C         frestore fp_cir+$2(a6)

RomBug: dx &.fp0
[reg] - 40000000C90FCF80DC337000 3.14159

RomBug: .fp1 0x40000000C90FCF80DC337000
RomBug: .
dn:00000000 00002700 00000002 00020000 00010002 00000001 FFFFE000 00004C00
an:00015100 FFFFE2800 00010000 00015100 0005C00 000150F8 00010000 000150B4
Using RomBug

Instruction Disassembly Memory Display

In the instruction disassembly display format, conditional instructions may be followed with a hyphen, followed by a right angle bracket (->) indicator. If -> is present, the instruction performs its TRUE operation, otherwise the instruction performs the FALSE operation. The appropriate condition code register is examined to determine which case the processor will perform.

Floating point conditional instructions use the condition portion of the 68881 FPSR register; the others use the processor CC register.

The following conditional instruction categories use this feature:

- **Bcc** Branch on condition
- **DBcc** Decrement and branch on condition
- **Scce** Set according to condition
- **TRAPcc** Trap on condition
- **FBcc** Branch on floating condition
Chapter 3: 68xxx Processors

FScc Set according to floating condition
FDBcc Decrement and branch on floating condition
FTRAPcc Trap on floating condition

Example

RomBug: di Main instruction disassembly

Main                >6002             bra.b initspu
Main+$2             >4E71             nop
initspu             >4E550000         link.w a5,#$0
initspu+$4          >48E7C0E0         movem.l d0-d1/a0-a2,-(a7)
initspu+$8          >2440             movea.l d0,a2
initspu+$A           >518F            subq.l #$8,a7
initspu+$C           >2D4A0004         move.l a2,$4(a6)
initspu+$10          >257C00001000088 move.l #$1000,$88(a2)
initspu+$18          >42AE0000         clrl $0(a6)
initspu+$1C           >7208            moveq.l #$8,d1
initspu+$1E           >41EE0000A        lea.l $A(a6),a0
initspu+$22           >2008            move.l a0,d0
initspu+$24           >61FF0000968     bsr.l clearmem
initspu+$2A           >303C7FFF         move.w #$7FFF,d0
initspu+$2E           >BE004F0000A      bfin.l $A(a6){$1:$F}
initspu+$34           >08AE0001000C   bclr.b #$1,$C(a6)

Floating Point Memory Display

The following is an example of floating point memory displays:

dis: df 20200
$00020200    - 40490FDB 3.141592741012573

dis: dd 20000
$00020000    - 400921FB54442D18 3.141592653589793

dis: dx 20100
$00020100    - 40000000C90FDBA22168C235 3.141592653589793
Setting and Displaying Debug Options

Use the \( o \) command to display and change the debugger modes. To display available options, use \( o? \). The following examples show the use of the \( o \) command with each of its options:

RomBug: \( o \) display option settings

Show control regs ON, Show FPU registers OFF

Show MMU registers off, Hexformat = 0x, MPU type = 68020/68881, Input radix = 16

RAM (hard) breakpoints

RomBug: \( o \) option help

\( a \) - Toggle control register display (68010/68020/68030/68040/683xx)

\( b<n> \) - Numeric input base radix

OS-9: \( c<n> \) - Set MPU type to \( <n> \) (68000, etc.), FPCP type to 6888<f>

\( d \) - Toggle FPCP decimal register display

OS-9: \( e<addr> \) - Display exception frame (default \( <addr> \) is .a7)

\( f \) - Toggle FPCP register display

\( m \) - Toggle MMU register display

\( r \) - Toggle rom type (soft) or ram type (hard) breakpoints

OS-9: \( x \) - Toggle disassembly hex output format

\( v \) - Display vectors being monitored

\( v[-][s|u][d]<n> [<m>] \) - Monitor exception vector ('-' to restore vector)

's' system state only, 'u' user state only,
'd' display only, \( <n> \) vector number in decimal,
\(<m> \) upper limit vector number in decimal

\( v? \) - Display all exception vector values

RomBug:.

\( \)dn:00000000 00020004 000DABC8 000DABBA 00000000 00000000 00840080
\( an \) 0000467C 000EC800 00004684 00004300 000ED960 000FE14C 00004300 00004300
\( pc:0000714A \) sr:2719 (--SI-7XN--C)t:OFF msp:EC57A71C usp:00013E7C  ^isp^ 0x000714A  >202C02E0  move.l 736(a4),d0

RomBug: \( o a \) toggle control registers

Show control regs ON, Show 68881 registers OFF

Hexformat = 0x, MPU type = 68020/68881, Input radix = 16

Ram (hard) breakpoints

RomBug:.

\( \)vbr:00000000 sfc:7=CS dfc:7=CS cacr:0=.. caar:D0062001
\( dn:00000000 00020004 000DABC8 000DABBA 00000000 00000000 00840080 \)
an: 0000467C 000EC800 00004684 00004300 000ED960 000FE14C 00004300 00004300
pc: 0000714A sr: 2719 (--SI--7XN--) t: OFF msp: EC57A71C usp: 00013E7C ^isp^ 0x0000714A > 202C02E0 move.l 736(a4), d0
RomBug: ob10 change input radix to base 10
Show control regs ON, Show 68881 registers OFF
Hexformat = 0x, MPU type = 68020/68881, Input radix = 10
Ram (hard) breakpoints
RomBug: d 10 I see we’re not foolin’
0x0000000A - 1E38FFF0 1E3EFFF0 1E440000 041E0000 .8.p.>.p.D......
RomBug: ov display currently monitored vectors
RomBug: ov 10 monitor vector 10
10 28 A-Line stop on supervisor/user state
RomBug: ov 4 7 monitor vector 4 through 7
4 10 Illegal Instruction stop on supervisor/user state
5 14 Zero Divide stop on supervisor/user state
6 18 CHK, CHK2 Instruction stop on supervisor/user state
7 1C cpTRAPcc, TRAPcc, TRAPV Instruction stop on supervisor/user state
RomBug: ov-4 10 quit monitoring vector 4 through 10
4 10 Illegal Instruction <not monitored>
5 14 Zero Divide <not monitored>
6 18 CHK, CHK2 Instruction <not monitored>
7 1C cpTRAPcc, TRAPcc, TRAPV Instruction <not monitored>
RomBug: ov 31 7C Level 7 Interrupt Autovector stop on supervisor/user state
RomBug: ovsd255 monitor and display supervisor state vector 255
255 3F User defined 255 display on supervisor state
RomBug: ovd254 monitor and display user state vector 254
254 3F8 User defined 254 display on user state
RomBug: ovd253 monitor and display vector 253
253 3F4 User defined 253 display on supervisor/user state
RomBug: ov
31 7C Level 7 Interrupt Autovector stop on supervisor/user state
253 3F4 User defined 253 display on supervisor/user state
254 3F8 User defined 254 display on user state
255 3FC User defined 255 display on supervisor state
RomBug: `ov-253 255` **quit monitoring vector 253 through 255**

- 253 3F4 User defined 253 <not monitored>
- 254 3F8 User defined 254 <not monitored>
- 255 3FC User defined 255 <not monitored>

RomBug: `ov`

- 31 7C Level 7 Interrupt Autovector stop on supervisor/user state

RomBug:
The following is provided in this section for the Pentium and 80x86 processors:
- o Options
Commands
Supported Registers
Display Information
Change Machine Registers
Instruction Disassembly Memory Display
Setting and Displaying Debug Options
### -o Options

The -o options identified in the following table are supported.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Display registers upon monitored exception</td>
</tr>
<tr>
<td>a</td>
<td>Show all CPU registers</td>
</tr>
<tr>
<td>d</td>
<td>Toggle FPCP decimal register display</td>
</tr>
<tr>
<td>f</td>
<td>Toggle FPCP register display</td>
</tr>
<tr>
<td>k[&lt;addr&gt;]</td>
<td>Remove watch point at &lt;addr&gt;</td>
</tr>
<tr>
<td>k*</td>
<td>Remove all watch points</td>
</tr>
<tr>
<td>p[&lt;M&gt;]&lt;port_num&gt;</td>
<td>Change I/O port contents at &lt;port_num&gt;</td>
</tr>
<tr>
<td></td>
<td>Change mode</td>
</tr>
<tr>
<td></td>
<td>Change word lengths (default is byte)</td>
</tr>
<tr>
<td></td>
<td>Change long lengths</td>
</tr>
<tr>
<td></td>
<td>No echo when changing</td>
</tr>
<tr>
<td></td>
<td>Change at odd port</td>
</tr>
<tr>
<td></td>
<td>Change at even port</td>
</tr>
<tr>
<td></td>
<td>Change prompt controls:</td>
</tr>
<tr>
<td></td>
<td>Move to next location</td>
</tr>
<tr>
<td></td>
<td>Move to previous location</td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>Move to next location and display</td>
</tr>
<tr>
<td>&lt;num&gt;</td>
<td>Store new value and move to next</td>
</tr>
<tr>
<td></td>
<td>Exit port change mode</td>
</tr>
<tr>
<td>wd{&lt;mode&gt;.&lt;size&gt;}&lt;addr&gt;</td>
<td>Set data watch point for &lt;addr&gt;</td>
</tr>
<tr>
<td>wi&lt;addr&gt;</td>
<td>Set instruction watch point for &lt;addr&gt;</td>
</tr>
<tr>
<td>w</td>
<td>Display watch points</td>
</tr>
</tbody>
</table>
Commands

Pentium and 80x86-specific command information is provided in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| e       | Enable/disable monitoring of processor-specific default exception vectors. Default vector numbers for exceptions are:  
          Invalid opcode= 6  
          Double fault= 8  
          General protection= D |

Supported Registers

The following table defines registers supported. Expressions and the $v$ command use the register names listed in the following table to obtain the value of a register.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.eax</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.ebx</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.ecx</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.edx</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.esi</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.edi</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.ebp</td>
<td>General purpose register</td>
</tr>
<tr>
<td>.esp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>.eip</td>
<td>Instruction pointer register</td>
</tr>
<tr>
<td>.eflags</td>
<td>Status register</td>
</tr>
<tr>
<td>.cs</td>
<td>Code segment register</td>
</tr>
<tr>
<td>.ds</td>
<td>Data segment register</td>
</tr>
<tr>
<td>Register</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>.es</td>
<td>Data segment register</td>
</tr>
<tr>
<td>.fs</td>
<td>Data segment register</td>
</tr>
<tr>
<td>.gs</td>
<td>Data segment register</td>
</tr>
<tr>
<td>.ss</td>
<td>Stack segment register</td>
</tr>
<tr>
<td>.gdtr</td>
<td>Base register for global descriptor table</td>
</tr>
<tr>
<td>.idtr</td>
<td>Base register for interrupt descriptor table</td>
</tr>
<tr>
<td>.ldtr</td>
<td>Selector register for local descriptor table</td>
</tr>
<tr>
<td>.tr</td>
<td>Task State Segment (TSS) register</td>
</tr>
<tr>
<td>.esp0 -</td>
<td>Privilege level 0-2 registers</td>
</tr>
<tr>
<td>.esp2</td>
<td></td>
</tr>
<tr>
<td>.ss0 - .ss2</td>
<td>Stack pointer for privilege level 0-2</td>
</tr>
<tr>
<td>.dr0 -.dr3</td>
<td>Debug registers</td>
</tr>
<tr>
<td>.dr6</td>
<td>Debug registers</td>
</tr>
<tr>
<td>.dr7</td>
<td>Debug registers</td>
</tr>
<tr>
<td>.cr0 - .cr3</td>
<td>Control registers</td>
</tr>
<tr>
<td>.fcr</td>
<td>Floating point control register</td>
</tr>
<tr>
<td>.ftw</td>
<td>Floating point tag register</td>
</tr>
<tr>
<td>.fsr</td>
<td>Floating point status register</td>
</tr>
<tr>
<td>.st0 - .st7</td>
<td>Floating point registers</td>
</tr>
</tbody>
</table>

**Display Information**

The following register information displays after an `oa` command:

```
ss0 : 0038   ss1 : 0038   ss2 : 0038
esp0: 00108000 esp1: 00108000 esp2: 00108000
```
cr0: 00000000 (--------------------------)
cr1: 00000000  cr2: 00000000  cr3: 00000000
gdtr: ffff0010002c  idtr: ffff00100c3c
eax: 00103F40  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000006 (--00----------P--)  
0x0000B72E  >8B5128  mov.l $28(%ecx),%edx

The eflags status control register value is interpreted as follows:

<table>
<thead>
<tr>
<th>eflags:00000006 (--00----------P--)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C = Carry bit set</td>
</tr>
<tr>
<td>Reserved, always set</td>
</tr>
<tr>
<td>P = Even parity</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>A = Auxiliary carry bit set</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>Z = Zero flag set</td>
</tr>
<tr>
<td>S = Sign flag set</td>
</tr>
<tr>
<td>T = Trap enable flag</td>
</tr>
<tr>
<td>I = Interrupt enable flag</td>
</tr>
<tr>
<td>D = Direction flag</td>
</tr>
<tr>
<td>O = Overflow bit set</td>
</tr>
<tr>
<td>I/O Privilege Level: 0 = 00, 1 = 01</td>
</tr>
<tr>
<td>2 = 10, 3 = 11</td>
</tr>
<tr>
<td>N = Nested task bit set</td>
</tr>
<tr>
<td>R = Restart flag set</td>
</tr>
<tr>
<td>V = Virtual 8086 mode</td>
</tr>
</tbody>
</table>

The cr0 register value for the 80386 is interpreted as follows:

G-------------------------XTEMP
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |

The cr0 register value for the 80486 is interpreted as follows:
The eight floating point registers are displayed in either hex or decimal form depending on the floating point register display option setting. The following is the hex display provided by the `of` command:

```assembly
eax: 00103F40  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000006 (---00----------P--)
st0: <Empty>  st4: <Empty>
st1: <Empty>  st5: <Empty>
st2: <Empty>  st6: <Empty>
st3: <Empty>  st7: <Empty>
fcr: 0F7F ZXPUOZDI  ftw: FFFF  fsr: 0000 (--00-0-------)
0x0000B72E >8B5128  mov.l $28(%ecx),%edx
```
If the `od` command is used, the registers are in the following decimal format:

```
RomBug: od
MPU type = 80386/80387, Input radix = 16
Ram (hard) breakpoints
Show all cpu regs OFF
Show 80387 registers ON in hex
RomBug: .st0 3.141592653589793
eax: 0021F468   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C edl: 00101858   ebp: 0011DBA4   esp: 0011DB90
eip: 0000B72E  eflags: 00000002 (--00-----------)
st0: 3.141592653589793  st4: <Empty>
st1: <Empty>  st5: <Empty>
st2: <Empty>  st6: <Empty>
st3: <Empty>  st7: <Empty>
fcr: 0F7F ZXPUOZDI  ftw: FFFF  fsr: 0000 (--000-0---------)
```

The floating point status registers display the following:

```
fcr:0F7F ZXPUDZDI floating point control register
ftw:FFFE floating point tag word register
fsr:0000 floating point status register
(--000-0--------)FPSR interpretation bits
```

The field to the right of the `fcr` register indicates exceptions that occurred during a floating point operation. These fields are interpreted as follows:

```
fcr: 0F7F NX------
I = Invalid operation
D = Denormal operand
Z = Zero divide
O = Overflow
U = Underflow
P = Precision
```

**80387 Floating point mask bits**

- Precision of significand: S=24 bits, D=53 bits, X=64 bits
- Rounding control:
  - N = Round toward nearest or even
  - - = Round toward negative infinity
  - + = Round toward positive infinity
  - 0 = Truncate toward zero
The ftw register provides an interpretation of the contents of the floating point registers: st0 - st7. Each digit in the ftw display directly maps to the contents of two registers in the following manner:

```
ftw: FFFF
    st0, st1
    st2, st3
    st4, st5
    st6, st7
```

content interpretation:
- 01 = Zero
- 00 = Valid
- 10 = Invalid or infinity
- 11 = Empty

For example, the following ftw display indicates that st7 has valid contents and that st0 through st7 are empty:

```
ftw: 3FFF
```

If any of the floating point registers have contents that are invalid or infinity, they are displayed as such:

```
st1:<Invalid or Infinity>
```

The register status bits indicate a variety of floating point conditions:

```
(--000-0--------)
    I = Invalid operation exception detected
    D = Denormal operand exception detected
    Z = Zero divide exception detected
    O = Overflow exception detected
    U = Underflow exception detected
    P = Precision exception detected
    S = Stack fault (over/underflow of the accumulator stack)
    E = Exception summary status
    C = Floating point condition code (C0).
        Maps to "CF" (carry flag) in eflag reg.
    L = Floating point condition code (Cl)
    P = Floating point condition code (C2).
        Maps to "PF" (parity flag) in eflags reg.
    B = Exception summary status (8087 compatibility)
```
A full register display example follows:

```
ss0  : 0038  ssi : 0038  ss2  : 0038
esp0: 00108000 esp1: 00108000 esp2: 00108000
cr0: 00000000 (--------------------------)
crl: 00000000 cr2: 00000000 cr3: 00000000
gdtr: ffff0010002c  idtr: ffff001000c3c
eax: 00103F40  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000006 (--00----------P--)
st0: <Empty>  st4: <Empty>
st1: <Empty>  st5: <Empty>
st2: <Empty>  st6: <Empty>
st3: <Empty>  st7: <Empty>
fcr: 0F7F ZXPUOZDI  ftw: FFFF  fsr: 0000 (--000-0--------)
0xB72E  >885128  mov.l $28(%ecx),%edx
```

Change Machine Registers

RomBug: of

MPU type = 80386/80387, Input radix = 16

Ram (hard) breakpoints

Show all cpu regs OFF

Show 80387 registers ON in hex

RomBug: .

eas: 0021F468  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000000 (---00--------)
st0: <Empty>  st4: <Empty>
st1: <Empty>  st5: <Empty>
st2: <Empty>  st6: <Empty>
st3: <Empty>  st7: <Empty>
fcr: 0F7F ZXPUOZDI  ftw: FFFF  fsr: 0000 (--000-0--------)
0x000B72E  >885128  mov.l $28(%ecx),%edx

RomBug: .eax 100

RomBug: .

eas: 00000100  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000000 (---00--------)
st0: <Empty>  st4: <Empty>
Using RomBug

st1: <Empty>                st5: <Empty>
st2: <Empty>                st6: <Empty>
st3: <Empty>                st7: <Empty>
fcr: 0F7F ZXPUOZDI   ftw: FFFF   fsr: 0000 (--000-0--------)
0xB72E              >8B5128           mov.l $28(%ecx),%edx
RomBug:  eax .ecx+.ebp
RomBug:  

eax: 021F468   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
eip: 0000B72E   eflags: 00000002 (--000-----------)
st0: <Empty>                st4: <Empty>
st1: <Empty>                st5: <Empty>
st2: <Empty>                st6: <Empty>
st3: <Empty>                st7: <Empty>
fcr: 0F7F ZXPUOZDI   ftw: FFFF   fsr: 0000 (--000-0--------)
xB72E              >8B5128           mov.l $28(%ecx),%edx
RomBug:  cd
MPU type = 80386/80387, Input radix = 16
Ram (hard) breakpoints
Show all cpu regs OFF
Show 80387 registers ON in hex
RomBug:  .st0 3.141592653589793

eax: 021F468   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
eip: 0000B72E   eflags: 00000002 (--000-----------)
st0: 3.141592653589793    st4: <Empty>
st1: <Empty>                st5: <Empty>
st2: <Empty>                st6: <Empty>
st3: <Empty>                st7: <Empty>
fcr: 0F7F ZXPUOZDI   ftw: FFFF   fsr: 0000 (--000-0--------)
0xB72E              >8B5128           mov.l $28(%ecx),%edx
RomBug:
Instruction Disassembly Memory Display

In the instruction disassembly display format, the \texttt{Jcc} conditional instructions may be followed with a hyphen, followed by a right angle bracket (\texttt{-\rangle}) indicator. If \texttt{-\rangle} is present, the instruction performs its \texttt{TRUE} operation, otherwise the instruction performs the \texttt{FALSE} operation. The appropriate condition code register is examined to determine the case the processor performs.

Example

\begin{verbatim}
RomBug: di main instruction disassembly
main                >60               pusha
main+$1             >8D6C2408      lea ss:$8(%esp),%ebp
main+$5             >8D6424FC      lea ss:$fc(%esp),%esp
main+$9             >8DB3061B0000   lea.l $1e06(%ebx),%esi
main+$F             >89F0             mov %esi,%eax
main+$11            >E8C7B2000       call setjmp
main+$16            >8945F4           mov.l %eax,ss:$f4(%ebp)
main+$19            >3D0000000000    cmp #$0,%eax
main+$1E             >7415             jz main+$35->
main+$20             >6A01             push #$1
main+$22             >8B45F4           mov.l ss:$f4(%ebp),%eax
main+$25             >E8B5E8FFFFFF    call put_exception
main+$2A             >8D642404      lea ss:$4(%esp),%esp
main+$2E             >29FF             sub %edi,%edi
main+$30             >E92F010000       jmp main+$164
main+$35             >E8F1EBFFFFFF    call get_vectors
\end{verbatim}

\begin{verbatim}
RomBug: di main 5 disassemble 5 instructions
main                >60               pusha
main+$1             >8D6C2408      lea ss:$8(%esp),%ebp
main+$5             >8D6424FC      lea ss:$fc(%esp),%esp
main+$9             >8DB3061B0000   lea.l $1e06(%ebx),%esi
main+$F             >89F0             mov %esi,%eax
\end{verbatim}

Floating Point Memory Displays

The following is an example of floating point memory displays:

\begin{verbatim}
trace: dl 120400
$00120400 - DA0F4940 65657266 182D4454 FB210940 Z.I@eerf.-DT{!.@
\end{verbatim}
Setting and Displaying Debug Options

Use the o command to display and change the debugger modes. To display available options, use o?. The following examples show the use of the o command with each of its options:

RomBug: o
Input radix = 16
Ram (hard) breakpoints
Show all cpu regs OFF
Watch execution SLOW
Show 80387 registers OFF
RomBug: o?
RomBug Options:

b<n>           Numeric input base radix
r              Use rom type (soft) breakpoints
v              Display vectors being monitored
v[-][s|u][d]<n> [<m>] Monitor exception vector ('-' to restore vector)
                  's' system state only,
                  'u' user state only
                  'd' display only, <m> range of vectors
v?             Display all exception vector values

80x86 Options:
.
      Display registers upon monitored exception
a
      Show all cpu registers
d      Toggle FPCP decimal register display
f      Toggle FPCP register display
k[<addr>]      Remove watch point at <addr>
k*      Remove all watch points
p[<M>]<port_num> Change i/o port contents at <port_num>
      <M>  change mode
Chapter 4: Pentium and 80x86 Processors

w  change word lengths (default is byte)
l  change long lengths
n  no echo when changing
o  change at odd port
e  change at even port
change prompt controls:
+  move to next location
-  move to previous location
<CR>  move to next location & display
<num>  store new value & move to next
.
exit port change mode

wd{<mode>.<size>}<addr>  Set data watch point for addr
wi<addr>                 Set instruction watch point for addr
w                         Display watch points

RomBug: .

eax: 0021F468  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000002 (---00-------------)
0xB72E              >8B5128           mov.l $28(%ecx),%edx

RomBug: OA

MPU type = 80386/80387, Input radix = 16
Ram (hard) breakpoints
Show all cpu regs ON
Show 80387 registers OFF
RomBug: .

ss0 : 0038  ss1 : 0038  ss2 : 0038
esp0: 00108000  esp1: 00108000  esp2: 00108000
cr0: 00000000 (--------------------------------)
cr1: 00000000  cr2: 00000000  cr3: 00000000
gdtr: ffff0010002c  idtr: ffff00100c3c
eax: 0021F468  ebx: 00100000  ecx: 001018C4  edx: 00103F40
esi: 0010188C  edi: 00101858  ebp: 0011DBA4  esp: 0011DB90
eip: 0000B72E  eflags: 00000002 (---00-------------)
0xB72E              >8B5128           mov.l $28(%ecx),%edx

RomBug: D 10 1
Using RomBug

0x10 - C83800F0 54FF00F0 745700F0 C83800F0 H8.pT..ptW.pH8.p

dis: ov

RomBug: ov 10

top on supervisor/user state

RomBug: ov 4 7

stop on supervisor/user state

RomBug: ov

stop on supervisor/user state

RomBug: ov-4 10

stop on supervisor/user state

RomBug: ov

<not monitored>

RomBug: ovsd40

display on supervisor state

RomBug: ovud41

display on user state

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

<not monitored>

RomBug: ov

Display on supervisor state

RomBug: ov

Display on user state

RomBug: ov

Display on supervisor state

RomBug: ov

Display on user state
RomBug: ov-40 45

40 100 User defined $40 <not monitored>
41 104 User defined $41 <not monitored>
42 108 User defined $42 <not monitored>
43 10C User defined $43 <not monitored>
44 110 User defined $44 <not monitored>
45 114 User defined $45 <not monitored>

RomBug: ov
This chapter discusses PowerPC processors. The supported PowerPC processors include the 403 (GA, GB, and GC), 405GB, 505, 601, 602, 603, 604, 8xx, and 82xx.

The following sections are included in this chapter:

- `o Options`
- `Commands`
- `Supported Registers`
- `Display Information`
- `Change Machine Registers`
- `Instruction Disassembly Memory Display`
- `Floating Point Memory Display`
- `Setting and Displaying Debug Options`
-o Options

-o options identified in the following table are supported by the PowerPC version of RomBug.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display registers upon monitored exception</td>
</tr>
<tr>
<td>a</td>
<td>Toggle control registers</td>
</tr>
<tr>
<td>d</td>
<td>Toggle FP decimal register display</td>
</tr>
<tr>
<td>f</td>
<td>Toggle FP register display</td>
</tr>
<tr>
<td>k[d</td>
<td>i][&lt;addr&gt;]</td>
</tr>
<tr>
<td></td>
<td>dKill data watch point</td>
</tr>
<tr>
<td></td>
<td>iKill instruction watch point</td>
</tr>
<tr>
<td>&lt;&lt;addr&gt;&gt;</td>
<td>If specified, checks that the address given is the</td>
</tr>
<tr>
<td></td>
<td>same as the one set before deleting it</td>
</tr>
<tr>
<td>m</td>
<td>Toggle MMU register display</td>
</tr>
<tr>
<td>tw</td>
<td>Trace over a set watch point trigger.</td>
</tr>
<tr>
<td>wd{&lt;mode&gt;}&lt;addr&gt;</td>
<td>Set data watch point for &lt;addr&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;mode&gt; access mode</td>
</tr>
<tr>
<td>r</td>
<td>read access</td>
</tr>
<tr>
<td>w</td>
<td>write access</td>
</tr>
<tr>
<td>rw</td>
<td>read/write access (default)</td>
</tr>
<tr>
<td>wd</td>
<td>Show data watch point</td>
</tr>
<tr>
<td>wi&lt;addr&gt;</td>
<td>Set instruction watch point for addr</td>
</tr>
<tr>
<td>w{i}</td>
<td>Show instruction watch point</td>
</tr>
</tbody>
</table>
Commands

PowerPC-specific command information is provided in the following table.

Table 5-2. Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Enable/disable monitoring of processor-specific default exception vectors. Default vector numbers for exceptions are: Machine check = 2, Data access = 3, Instruction access = 4, Alignment = 6, Machine program = 7</td>
</tr>
</tbody>
</table>

Supported Registers

The following tables define user and supervisor registers specific to each processor. Table 5-3, PowerPC User Registers identifies user registers common to the supported processors (exceptions are noted in the table’s legend). Table 5-4, 601-Specific User Registers identifies user registers unique to the 601 processor. Table 5-5, Supervisor Registers Sorted Numerically identifies supervisor registers for the supported processors (exceptions are noted in the table’s legend), sorted numerically by register number.

Expressions and the v command use the register names as shown in the register tables to obtain the value of a register.

User Registers

Table 5-3. PowerPC User Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0 - r31 †</td>
<td>gpr0 - gpr31</td>
<td>General purpose registers</td>
</tr>
<tr>
<td>cr</td>
<td>-</td>
<td>Condition register</td>
</tr>
<tr>
<td>f0 - f31 *(1)</td>
<td>fpr0 - fpr31</td>
<td>Floating point registers</td>
</tr>
<tr>
<td>fpscr *(1)</td>
<td>-</td>
<td>Floating point status and control register</td>
</tr>
<tr>
<td>xer</td>
<td>spri</td>
<td>Integer exception register</td>
</tr>
<tr>
<td>lr</td>
<td>spri8</td>
<td>Link register</td>
</tr>
<tr>
<td>pc</td>
<td>-</td>
<td>Program counter</td>
</tr>
</tbody>
</table>
Using RomBug

† Note that r# in PowerPC specifies general purpose registers and that the rr# specifies relocation registers.
*(1) Not available on 403, 405, or 8xx processors
*(2) Except 602 processor
*(3) Not available on 403, 601 processors

Table 5-3. PowerPC User Registers (Continued)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msr</td>
<td>-</td>
<td>Machine status register</td>
</tr>
<tr>
<td>sp</td>
<td>spr1 *(2)</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>ctr</td>
<td>spr9</td>
<td>Count register</td>
</tr>
<tr>
<td>tbl * (3)</td>
<td>spr268</td>
<td>Time base lower (read only)</td>
</tr>
<tr>
<td>tbu * (3)</td>
<td>spr269</td>
<td>Time base upper (read only)</td>
</tr>
</tbody>
</table>

Table 5-4. 601-Specific User Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq</td>
<td>spr0</td>
<td>MQ register</td>
</tr>
<tr>
<td>rtcu</td>
<td>spr4</td>
<td>RTC upper register (read only)</td>
</tr>
<tr>
<td>rtcl</td>
<td>spr5</td>
<td>RTC lower register (read only)</td>
</tr>
</tbody>
</table>

Supervisor Registers

Table 5-5. Supervisor Registers Sorted Numerically

<table>
<thead>
<tr>
<th>SPR #</th>
<th>403 SPR Name</th>
<th>505 SPR Name</th>
<th>601 SPR Name</th>
<th>602 SPR Name</th>
<th>82xx/603 SPR Name</th>
<th>604 SPR Name</th>
<th>8xx SPR Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>xer</td>
<td>xer</td>
<td>xer</td>
<td>xer</td>
<td>xer</td>
<td>xer</td>
<td>xer</td>
</tr>
<tr>
<td>SPR #</td>
<td>403 SPR Name</td>
<td>505 SPR Name</td>
<td>601 SPR Name</td>
<td>602 SPR Name</td>
<td>82xx/ 603 SPR Name</td>
<td>604 SPR Name</td>
<td>8xx SPR Name</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>8</td>
<td>lr</td>
<td>lr</td>
<td>lr</td>
<td>lr</td>
<td>lr</td>
<td>lr</td>
<td>lr</td>
</tr>
<tr>
<td>9</td>
<td>ctr</td>
<td>ctr</td>
<td>ctr</td>
<td>ctr</td>
<td>ctr</td>
<td>ctr</td>
<td>ctr</td>
</tr>
<tr>
<td>18</td>
<td>dsisr</td>
<td>dsisr</td>
<td>dsisr</td>
<td>dsisr</td>
<td>dsisr</td>
<td>dsisr</td>
<td>dsisr</td>
</tr>
<tr>
<td>19</td>
<td>dar</td>
<td>dar</td>
<td>dar</td>
<td>dar</td>
<td>dar</td>
<td>dar</td>
<td>dar</td>
</tr>
<tr>
<td>20</td>
<td>rtcu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(write only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>rtcl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(write only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>dec</td>
<td>dec</td>
<td>dec</td>
<td>dec</td>
<td>dec</td>
<td>dec</td>
<td>dec</td>
</tr>
<tr>
<td>25</td>
<td>sdr1</td>
<td>sdr1</td>
<td>sdr1</td>
<td>sdr1</td>
<td>sdr1</td>
<td>sdr1</td>
<td>sdr1</td>
</tr>
<tr>
<td>26</td>
<td>srr0</td>
<td>srr0</td>
<td>srr0</td>
<td>srr0</td>
<td>srr0</td>
<td>srr0</td>
<td>srr0</td>
</tr>
<tr>
<td>27</td>
<td>srr1</td>
<td>srr1</td>
<td>srr1</td>
<td>srr1</td>
<td>srr1</td>
<td>srr1</td>
<td>srr1</td>
</tr>
<tr>
<td>80</td>
<td>eie</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>eie</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>eid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>eid</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>nri</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nri</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>cmpa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cmpa</td>
</tr>
<tr>
<td>145</td>
<td>cmpb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cmpb</td>
</tr>
<tr>
<td>146</td>
<td>cmpc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cmpc</td>
</tr>
<tr>
<td>147</td>
<td>cmpd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cmpd</td>
</tr>
<tr>
<td>148</td>
<td>ecr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>icr</td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>der</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>der</td>
<td></td>
</tr>
</tbody>
</table>
Table 5-5. Supervisor Registers Sorted Numerically (Continued)

<table>
<thead>
<tr>
<th>SPR #</th>
<th>403 SPR Name</th>
<th>505 SPR Name</th>
<th>601 SPR Name</th>
<th>602 SPR Name</th>
<th>82xx/603 SPR Name</th>
<th>604 SPR Name</th>
<th>8xx SPR Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>counta</td>
<td></td>
<td></td>
<td></td>
<td>counta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>countb</td>
<td></td>
<td></td>
<td></td>
<td>countb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>cmpe</td>
<td></td>
<td></td>
<td></td>
<td>cmpe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>cmpf</td>
<td></td>
<td></td>
<td></td>
<td>cmpf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>cmpg</td>
<td></td>
<td></td>
<td></td>
<td>cmpg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>cmph</td>
<td></td>
<td></td>
<td></td>
<td>cmph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>lctr11</td>
<td></td>
<td></td>
<td></td>
<td>lctr11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>lctr12</td>
<td></td>
<td></td>
<td></td>
<td>lctr12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>ictrl</td>
<td></td>
<td></td>
<td></td>
<td>ictrl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>bar</td>
<td></td>
<td></td>
<td></td>
<td>bar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>272</td>
<td>sprg0</td>
<td>sprg0</td>
<td>sprg0</td>
<td>sprg0</td>
<td>sprg0</td>
<td>sprg0</td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>sprg1</td>
<td>sprg1</td>
<td>sprg1</td>
<td>sprg1</td>
<td>sprg1</td>
<td>sprg1</td>
<td></td>
</tr>
<tr>
<td>274</td>
<td>sprg2</td>
<td>sprg2</td>
<td>sprg2</td>
<td>sprg2</td>
<td>sprg2</td>
<td>sprg2</td>
<td></td>
</tr>
<tr>
<td>275</td>
<td>sprg3</td>
<td>sprg3</td>
<td>sprg3</td>
<td>sprg3</td>
<td>sprg3</td>
<td>sprg3</td>
<td></td>
</tr>
<tr>
<td>282</td>
<td>ear</td>
<td>ear</td>
<td>ear</td>
<td>ear</td>
<td>ear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>284</td>
<td>tbl</td>
<td>tbl</td>
<td>tbl</td>
<td>tbl</td>
<td>tbl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>285</td>
<td>tbu</td>
<td>tbu</td>
<td>tbu</td>
<td>tbu</td>
<td>tbu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>pvr</td>
<td>pvr</td>
<td>pvr</td>
<td>pvr</td>
<td>pvr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>528</td>
<td>ibat0u</td>
<td>ibat0u</td>
<td>ibat0u</td>
<td>ibat0u</td>
<td>ibat0u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>529</td>
<td>ibat0l</td>
<td>ibat0l</td>
<td>ibat0l</td>
<td>ibat0l</td>
<td>ibat0l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>530</td>
<td>ibat1u</td>
<td>ibat1u</td>
<td>ibat1u</td>
<td>ibat1u</td>
<td>ibat1u</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 5-5. Supervisor Registers Sorted Numerically (Continued)

<table>
<thead>
<tr>
<th>SPR #</th>
<th>403 SPR Name</th>
<th>505 SPR Name</th>
<th>601 SPR Name</th>
<th>602 SPR Name</th>
<th>82xx/ 603 SPR Name</th>
<th>604 SPR Name</th>
<th>8xx SPR Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>531</td>
<td>ibat1l</td>
<td>ibat1l</td>
<td>ibat1l</td>
<td>ibat1l</td>
<td>ibat1l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>532</td>
<td>ibat2u</td>
<td>ibat2u</td>
<td>ibat2u</td>
<td>ibat2u</td>
<td>ibat2u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>533</td>
<td>ibat2l</td>
<td>ibat2l</td>
<td>ibat2l</td>
<td>ibat2l</td>
<td>ibat2l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>534</td>
<td>ibat3u</td>
<td>ibat3u</td>
<td>ibat3u</td>
<td>ibat3u</td>
<td>ibat3u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>535</td>
<td>ibat3l</td>
<td>ibat3l</td>
<td>ibat3l</td>
<td>ibat3l</td>
<td>ibat3l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>536</td>
<td>dbat0u</td>
<td>dbat0u</td>
<td>dbat0u</td>
<td>dbat0u</td>
<td>dbat0u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>537</td>
<td>dbat0l</td>
<td>dbat0l</td>
<td>dbat0l</td>
<td>dbat0l</td>
<td>dbat0l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>538</td>
<td>dbat1u</td>
<td>dbat1u</td>
<td>dbat1u</td>
<td>dbat1u</td>
<td>dbat1u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>539</td>
<td>dbat1l</td>
<td>dbat1l</td>
<td>dbat1l</td>
<td>dbat1l</td>
<td>dbat1l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>540</td>
<td>dbat2u</td>
<td>dbat2u</td>
<td>dbat2u</td>
<td>dbat2u</td>
<td>dbat2u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>541</td>
<td>dbat2l</td>
<td>dbat2l</td>
<td>dbat2l</td>
<td>dbat2l</td>
<td>dbat2l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>542</td>
<td>dbat3u</td>
<td>dbat3u</td>
<td>dbat3u</td>
<td>dbat3u</td>
<td>dbat3u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>543</td>
<td>dbat3l</td>
<td>dbat3l</td>
<td>dbat3l</td>
<td>dbat3l</td>
<td>dbat3l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>560</td>
<td>ic_cst</td>
<td></td>
<td></td>
<td></td>
<td>ic_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>561</td>
<td>ic_adr</td>
<td></td>
<td></td>
<td></td>
<td>ic_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>562</td>
<td>ic_dat</td>
<td></td>
<td></td>
<td></td>
<td>ic_dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>563</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>564</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>565</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>566</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>567</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>568</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>569</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>570</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dc_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>630</td>
<td>dpdr</td>
<td></td>
<td></td>
<td></td>
<td>dpdr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>631</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dpdr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPR #</td>
<td>403 SPR Name</td>
<td>505 SPR Name</td>
<td>601 SPR Name</td>
<td>602 SPR Name</td>
<td>82xx/603 SPR Name</td>
<td>604 SPR Name</td>
<td>8xx SPR Name</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>638</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>immr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>784</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_ctr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>786</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_ap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>787</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_epn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>789</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_mi_twc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>790</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_rpn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>792</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_ctr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>793</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m_casid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>794</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_ap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>795</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_epn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>796</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m_tw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>797</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_twc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>798</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_rpn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>799</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m_tw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>816</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_dbcam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>817</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_dbram0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>818</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mi_dbram1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>824</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_dbcam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>825</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_dbram0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>826</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>md_dbram1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>952</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mmcr0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPR #</td>
<td>403 SPR Name</td>
<td>505 SPR Name</td>
<td>601 SPR Name</td>
<td>602 SPR Name</td>
<td>82xx/603 SPR Name</td>
<td>604 SPR Name</td>
<td>8xx SPR Name</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>953</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pmc1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>954</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pmc2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>955</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>959</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>976</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dmiss</td>
<td>dmiss</td>
<td></td>
</tr>
<tr>
<td>977</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dcmp</td>
<td>dcmp</td>
<td></td>
</tr>
<tr>
<td>978</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hash1</td>
<td>hash1</td>
<td></td>
</tr>
<tr>
<td>979</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hash2</td>
<td>hash2</td>
<td></td>
</tr>
<tr>
<td>980</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>esr</td>
<td>imiss</td>
<td>imiss</td>
</tr>
<tr>
<td>981</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dear</td>
<td>icmp</td>
<td>icmp</td>
</tr>
<tr>
<td>982</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>evpr</td>
<td>rpa</td>
<td>rpa</td>
</tr>
<tr>
<td>984</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tsr</td>
<td>tcr</td>
<td></td>
</tr>
<tr>
<td>986</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tcr</td>
<td></td>
<td>ibr</td>
</tr>
<tr>
<td>987</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pit</td>
<td>esasrr</td>
<td></td>
</tr>
<tr>
<td>988</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tbhi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>989</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tblo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>990</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>srr2</td>
<td></td>
<td>sebr</td>
</tr>
<tr>
<td>991</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>srr3</td>
<td></td>
<td>ser</td>
</tr>
<tr>
<td>1008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dbsr</td>
<td>hid0</td>
<td>hid0</td>
</tr>
<tr>
<td>1009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hid1</td>
<td>hid1</td>
</tr>
<tr>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dbcr</td>
<td>iabr</td>
<td>iabr</td>
</tr>
</tbody>
</table>
Table 5-5. Supervisor Registers Sorted Numerically (Continued)

<table>
<thead>
<tr>
<th>SPR #</th>
<th>403 SPR Name</th>
<th>505 SPR Name</th>
<th>601 SPR Name</th>
<th>602 SPR Name</th>
<th>82xx/603 SPR Name</th>
<th>604 SPR Name</th>
<th>8xx SPR Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1012</td>
<td>iac1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1013</td>
<td>iac2</td>
<td>dabr</td>
<td>dabr</td>
<td>dabr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1014</td>
<td>dac1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1015</td>
<td>dac2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1018</td>
<td>dccr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1019</td>
<td>iccr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1020</td>
<td>pbl1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1021</td>
<td>pbu1</td>
<td>sp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1022</td>
<td>pbl2</td>
<td>fpecr</td>
<td>lt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1023</td>
<td>pbu2</td>
<td>pir</td>
<td>pir</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Display Information**

Normal register display:

```
r0: 00FE8A60 00FE8A00 00029C40 00000000 00000000 00000000 00000000 00000000
r8: 00105B90 00FE8DB8 00000000 00000000 00000000 00000000 00000000 00000000
r16: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
pc: 0020F3D0 lr: 48003614 ctr: 00004400 mar: 000001030 (---C------ID----cr: 20000000 (---Z-----------------------------)
0x20F3D0 >7C0802A6 mflr r0
```
The `cr` value is interpreted as follows:

```
CR2 CR3 CR4 CR5 CR6 CR7
;r:20000000 (--Z--------)------

X = Floating Point Exception
E = Floating Point Enabled Exception
V = Floating Point Invalid Exception
O = Floating Point Overflow Exception
O = Summary Overflow
Z = Zero
P = Positive
N = Negative
```

CR2 through CR7: ----

```
O = Summary Overflow
Z = Zero
P = Positive
N = Negative
```

The `msr` value is interpreted as follows:

```
msr:0001030 (---C------ID----)
```

```
D = Data Address Translation Enable Bit
I = Instruction Address Translation Enable Bit
E = Exception Prefix Enable Bit
M = Floating Point Exception Mode 1 Bit
B = Debug Enable (403), Branch Taken Enable (603)
T = Single Step Trace Enable Bit
M = Floating Point Exception Mode 0 Exception Bit
C = Machine Check Enable Bit
F = Floating Point Enable Bit
P = Privilege Level Bit
X = External Exception (IRQ) Enable Bit
```

601 MMU registers:

```
sr0:E7F00000 E7F00001 E7F00002 E7F00003 E7F00004 E7F00005 E7F00006 E7F
sr8:E7F00008 E7F00009 E7F0000A E7F0000B E7F0000C E7F0000D E7F0000E E7F
bat:00000000 00000000 00000000 00000000 00000000 00000000 00000000 000
```
603 MMU registers:

dmiss: 43A67C43 imiss: 00244978 dcmp: 8000018E icmp: 80000180
hash1: 00FC9900 hash2: 00FD66C0 rpa: 0023F199 sdr1: 00FC0001
sr0: 20000003 sr1: 20000003 sr2: 20000003 sr3: 20000003
sr4: 20000003 sr5: 20000003 sr6: 20000003 sr7: 20000003
sr8: 20000003 sr9: 20000003 sr10: 20000003 sr11: 20000003
ibat0u: 00000000 ibat1u: 00000000 ibat2u: 00000000 ibat3u: 00000000
ibat0l: 00000000 ibat1l: 00000000 ibat2l: 00000000 ibat3l: 00000000
dbat0u: 00000000 dbat1u: 00000000 dbat2u: 00000000 dbat3u: 00000000
dbat0l: 00000000 dbat1l: 00000000 dbat2l: 00000000 dbat3l: 00000000

The following register information displays following an oa command:

srr0: 00102144 srr1: 00003030
sprg0: 0029C40 sprg1: 00FE8144 sprg2: 40000000 sprg3: 00FE8A60
dsisr: 40000000 dar: 43A67C43 ear: 00000000 sdr1: 00FC0001

601/603 floating point register dump:

f0: fff8000000000000 0000000000000000 0000000000000000 0000000000000000
f4: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f8: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f12: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f16: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f20: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f24: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f28: 0000000000000000 0000000000000000 0000000000000000 fff8000000000000
fpscr: 00000000 (----------------------------------)
0x20F3D0      >7C0802A6       mflr r0

If a decimal display is indicated, the registers appear in the following format:

f0: <NaN>  0  0  0
f4: 0  0  0  0
f8: 0  0  0  0
f12: 0  0  0  0
f16: 0  0  0  0
f20: 0  0  0  0
f24: 0  0  0  0
f28: 0  0  <NaN>  0
fpscr: 00000000 (----------------------------------)
0x20F3D0      >7C0802A6       mflr r0
PowerPC display (hex and decimal):

```
RomBug:  d 100400
$00100400  - 60A40000 80A50018 7CA903A6 4E800421 `$.%.|)`..&N...$0
28030000 4082FFCC 48000014 38000001
(....@..LH....8...
$00100420  - 7FE0F850 281F0000 4082FF7C 8002B210 .`xP(....@..|2.
$00100430  - 28000000 41820014 48008219 8062B208 (....A...H...b2.
$00100440  - 8002B538 90030002C 8002B208 60030000 ...8...,2.``...
$00100450  - 48000505 BBC10008 38210010 80010004 H...A..8!....
$00100460  - 7C0803A6 4E800200 9421FF88 8002B23C |..&N...!..x..2<
$00100540  - 28000040 40820008 48000028 8002B23C (....@@...H...(.2<
$00100480  - 30A00001 90A2B23C 38C00003 7C003030 0 ..."2<8...|..00
$00100490  - 38A28020 7CA50214 7C82929E 90650004 8". |%..|.).e.
$001004A0  - 38210008 4E800200 7C0802A6 90010004 8!..N...|.&...n
$001004B0  - 9421FF88 BFC10008 607F0000 609B0000 .!h?A...````````.
$001004C0  - 38000094 90010010 38610010 3882B208 8"....8a..8.2.
$001004D0  - 38A00000 480009AD 2B030000 41820008 8"...H...(....A...
$001004E0  - 4BFPPFC5 8082B208 80610010 38000000 K...2...a...8...-
$001004F0  - 5463F0BE 4800001C 60850000 30840004 Tcp>H...`````````````
```

Change Machine Registers

Examples

RomBug:  ...
```
  r0:00000000 00FA80B8 002D9C40 00000000 00004400 0030A6C4 0030A6CA
  r8:00030E78 002D9D88 00000003 000000DC 00000000 02113888 00FB3C00 00000000
  r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00009030
  pc:0020BA2C lr: 0020BA2C ctr: 00004400 msr: 00001030 (---C------ID----)
  cr:40000000 (-P-----------------------------------)
  f0: <NaN>       0       0       0
  f4: 0           0       0       0
  f8: 0           0       0       0
  f12: 0          0       0       0
  f16: 0          0       0       0
  f20: 0          0       0       0
  f24: 0          0       0       0
  f28: 0          0       0       0
  fpscr: 00000000 (----------------------------------)
```
```
RomBug: .  r4 100
RomBug:  ...
```
```
```
```
```
```
```
```
Using RomBug

RomBug:  .r4.r2+.r6
RomBug:  .
r0:00000000 00FE80B8 00029C40 00000000 00334304 00004400 0030A6C4 0030A6CA
r8:00030E78 00209D8 00000003 000000DC 00000000 00211388 00FE3C00 00000000
r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00009030
pc:0020BA2C lr: 0020BA2C ctr: 00004400 msr: 00001030 (---C------ID----)
cr:40000000 (---P----------------------------------)
f:0: <NaN>             0                  0                 0
f:4: 0                 0                  0                 0
f:8: 0                 0                  0                 0
f:12: 0                 0                  0                 0
f:16: 0                 0                  0                 0
f:20: 0                 0                  0                 0
f:24: 0                 0                  0                 0
f:28: 0                 0                  0                 0
fpscr: 00000000 (----------------------------------)
0x0020BA2C   >7FE00124         mtmsr r31

RomBug:  .f0 4
RomBug:  .
r0:00000000 00FE80B8 00029C40 00000000 00334304 00004400 0030A6C4 0030A6CA
r8:00030E78 00209D8 00000003 000000DC 00000000 00211388 00FE3C00 00000000
r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00009030
pc:0020BA2C lr: 0020BA2C ctr: 00004400 msr: 00001030 (---C------ID----)
cr: 40000000 (---P----------------------------------)
f:0: 4                 0                  0                 0
f:4: 0                 0                  0                 0
f:8: 0                 0                  0                 0
f:12: 0                 0                  0                 0
f:16: 0                 0                  0                 0
f:20: 0                 0                  0                 0
f:24: 0                 0                  0                 0
f:28: 0                 0                  0                 0
fpscr: 00000000 (----------------------------------)
0x0020BA2C   >7FE00124         mtmsr r31
Chapter 5: PowerPC Processors

RomBug: .f4 3.14159
RomBug: .
  r0:00000000 00FE80B8 00029C40 00000000 00334304 00004400 0030A6C4 0030A6CA
  r8:00030E78 0020B9D8 00000003 000000DC 00000000 00211388 00FE3C00 00000000
  r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  pc:0020BA2C lr: 0020BA2C ctr: 00004400 msr: 00001030 (--C------ID----)
  cr:40000000 (-P------------------------------)

  f0: 4                 0                  0                 0
  f4: 3.14159           0                  0                 0
  f8: 0                 0                  0                 0
  f12: 0                0                  0                 0
  f16: 0                0                  0                 0
  f20: 0                0                  0                 0
  f24: 0                0                  0                 0
  f28: 0                0                  0                 0

fpscr: 00000000 (-- )
  0x0020BA2C   >7FE00124         mtmsr r31
dis: dx &.f4
[reg] - 400921F9F01B866F 3.14159
dis: .f4 0x00000000c90f80dc80dc37000
RomBug: .
  r0:00000000 00FE80B8 00029C40 00000000 00334304 00004400 0030A6C4 0030A6CA
  r8:00030E78 0020B9D8 00000003 000000DC 00000000 00211388 00FE3C00 00000000
  r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  pc:0020BA2C lr: 0020BA2C ctr: 00004400 msr: 00001030 (--C------ID----)
  cr:40000000 (- P------------------------------)

  f0: 4                 0                  0                 0
  f4: 0.00026666e-309   0                  0                 0
  f8: 0                 0                  0                 0
  f12: 0                0                  0                 0
  f16: 0                0                  0                 0
  f20: 0                0                  0                 0
  f24: 0                0                  0                 0
  f28: 0                0                  0                 0

fpscr: 00000000 (-- )
  0x0020BA2C   >7FE00124         mtmsr r31
RomBug: cd

MPU type = 603, Input radix = 16
Ram (hard) breakpoints
Show control regs OFF, Show FP regs ON in hex
Show MMU regs OFF
RomBug:
Instruction Disassembly Memory Display

RomBug: di Main
main >7C0802A6 mflr r0
main+$4 >90010004 stw r0,4(r1)
main+$8 >9421FFEC stwu r1,-20(r1)
main+$C >BFC1000C stmw r30,12(r1)
main+$10 >3C000000 addis r0,r0,0
main+$14 >60003AB0 ori r0,r0,15072
main+$18 >7C620214 add r3,r2,r0
main+$1C >4800FAC1 bl __setjmp
main+$20 >2C030000 cmpwi cr0,r3,0
main+$24 >41820014 beq cr0,main+$38
main+$28 >38800001 addi r4,r0,1
main+$2C >4800A869 bl put_exception
main+$30 >3BE00000 addi r31,r0,0
main+$34 >4800019C b main+$1D0
main+$38 >48009F5D bl get_vectors
main+$3C >3C000000 addis r0,r0,0
dis: di main 5
main >7C0802A6 mflr r0
main+$4 >90010004 stw r0,4(r1)
main+$8 >9421FFEC stwu r1,-20(r1)
main+$C >BFC1000C stmw r30,12(r1)
main+$10 >3C000000 addis r0,r0,0

dis:

Floating Point Memory Display

The following is an example of a floating point memory display.

dis: df 20200
$00020200 - 40690FDB 3.141592741012573
dis: dd 20000
$00020000 - 400921FBS4442D18 3.141592653589793

Setting and Displaying Debug Options

Use the o command to display and change the debugger modes. To display available options, use o?. The following examples show the use of the o command with each of its options:

RomBug: o
MPU type = 601, Input radix = 16
Ram (hard) breakpoints
Show control regs OFF, Show FP regs OFF
Show MMU regs OFF

RomBug: o?
RomBug Options:

b<n> Numeric input base radix
r Use rom type (soft) breakpoints
s Toggle showing cpu registers during trace
v Display vectors being monitored
v[-][s|u][d]<n> [m>] Monitor exception vector (‘-’ to restore vector)
   ‘s’ system state only, ‘u’ user state only
   ‘d’ display only, <m> range of vectors
v? Display all exception vector values

PowerPC Options:
.
   Display registers upon monitored exception
a Toggle control registers
d Toggle FP decimal register display
f Toggle FP register display
k[d|i]<addr> Kill watch point
   d Kill data watch point †
   i Kill instruction watch point
   <addr> if specified, checks that the address
given is the same as the one set before
deleting it. <addr> MUST be specified on
CPU’s with multiple watch points
m Toggle MMU register display
tw Trace passed a set watch point trigger
wd{<mode>}<addr> Set data watch point for addr
   <mode> access mode, one of
   r read access
   w write access
   rw read/write access (default)
wd Show data watch point
wi<addr> Set instruction watch point for addr
w{i} Show instruction watch point

RomBug:
r0:00000001 0028625C 0028BFFC 00000001 00286298 002862A0 00284D9C
00000002
Using RomBug

r8: 00286288 00000000 00000000 00000000 00000000 00000000 001CE3AC 00000000
r16: 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24: 00000000 00000000 00000000 00000000 00000000 00000000 002862B0
pc: 001C66D4 lr: 001C65B0 ctr: 00000000 msr: 00009000 (X--C------------)
cr: 20000000 (--Z----------------------------------)
RomBug: oa
MPU type = 603, Input radix = 16
Ram (hard) breakpoints
Show control regs ON, Show FP regs OFF
Show MMU regs OFF
RomBug:
srr0: 001C66D4 srr1: 00089000
sprg0: 00200C00 sprg1: 00286288 sprg2: 00000000 sprg3: 0028625C
disr: 00000000 dar: 00000000 ear: 00000000 sdr1: 00FC0001
r0: 00000000 0028625C 0028BFFC 00000001 00286298 002862A0 00284D9C 00000002
r8: 00286288 00000000 00000000 00000000 00000000 00000000 001CE3AC 00000000
r16: 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24: 00000000 00000000 00000000 00000000 00000000 00000000 002862B0
pc: 001C66D4 lr: 001C65B0 ctr: 00000000 msr: 00009000 (X--C------------)
cr: 20000000 (--Z----------------------------------)
RomBug: of
MPU type = 603, Input radix = 16
Ram (hard) breakpoints
Show control regs ON, Show FP regs ON in hex
Show MMU regs OFF
RomBug:
srr0: 001C66D4 srr1: 00089000
sprg0: 00200C00 sprg1: 00286288 sprg2: 00000000 sprg3: 0028625C
disr: 00000000 dar: 00000000 ear: 00000000 sdr1: 00FC0001
r0: 00000000 0028625C 0028BFFC 00000001 00286298 002862A0 00284D9C 00000002
r8: 00286288 00000000 00000000 00000000 00000000 00000000 001CE3AC 00000000
r16: 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Chapter 5: PowerPC Processors

r24:00000000 00000000 00000000 00000000 00000000 00000000 002862B0
pc: 001C66D4 lr: 001C65B0 ctr: 00000000 mscr: 00009000 (X--C-----------)
cr: 20000000 (--Z-----------------------------------)
f0:ffe8000000000000 0000000000000000 0000000000000000 0000000000000000
f4:0000000000000000 0000000000000000 0000000000000000 0000000000000000
f8:0000000000000000 0000000000000000 0000000000000000 0000000000000000
f12:0000000000000000 0000000000000000 0000000000000000 0000000000000000
f16:0000000000000000 0000000000000000 0000000000000000 0000000000000000
f20:0000000000000000 0000000000000000 0000000000000000 0000000000000000
f24:0000000000000000 0000000000000000 0000000000000000 0000000000000000
f28:0000000000000000 0000000000000000 fff8000000000000 0000000000000000
fpscr: 00000000 (----------------------------------)
RomBug: om
MPU type = 603, Input radix = 16
Ram (hard) breakpoints
Show control regs ON, Show FP regs ON in hex
Show MMU regs ON
RomBug:
srr0: 001C66D4 srr1: 00089000
sprg0: 00200C00 sprg1: 00286288 sprg2: 20000000 sprg3: 0028625C
disr: 00000000 dar: 00000000 ear: 00000000 sdr1: 00FC0001
dmiss: 001C63C2 imiss: 001662C4 dcmp: FE9EC100 icmp: FE9EC100
hash1: 00FD3900 hash2: 00FC66C0 rpa: 0166199 sdr1: 00FC0001
sr0: 20F8F003 sr1: 20F8F003 sr2: 20F8F003 sr3: 20F8F003
sr4: 20F8F003 sr5: 20F8F003 sr6: 20F8F003 sr7: 20F8F003
sr8: 20F8F003 sr9: 20F8F003 sr10: 20F8F003 sr11: 20F8F003
sr12: 20F8F003 sr13: 20F8F003 sr14: 20F8F003 sr15: 20F8F003
ibat0u:00000000 ibat1u:00000000 ibat2u:00000000 ibat3u:00000000
ibat0l:00000000 ibat1l:00000000 ibat2l:00000000 ibat3l:00000000
dbat0u:00000000 dbat1u:00000000 dbat2u:00000000 dbat3u:00000000
dbat0l:00000000 dbat1l:00000000 dbat2l:00000000 dbat3l:00000000
r0:00000001 0028625C 00286298 002862A0 00284D9C
r1:00000002
r2:00286288 00000000 00000000 00000000 00000000 001CE3AC 00000000
r3:00000000
r4:00000000 00000000 00000000 00000000 00000000 00000000 00000000
r5:00000000 00000000 00000000 00000000 00000000 00000000 00000000
r6:00000000 00000000 00000000 00000000 00000000 00000000 00000000
r7:00000000 00000000 00000000 00000000 00000000 00000000 00000000
r8:00286288 00000000 00000000 00000000 00000000 001CE3AC 00000000
r9:00000000
r10:00000000 00000000 00000000 00000000 00000000 00000000 00000000
r11:00000000 00000000 00000000 00000000 00000000 00000000 00000000
r12:00000000 00000000 00000000 00000000 00000000 00000000 00000000

cr: 20000000 (--Z-----------------------------)

f0: fff8000000000000 0000000000000000 0000000000000000 0000000000000000
f4: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f8: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f12: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f16: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f20: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f24: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f28: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
fpscr: 00000000 (----------------------------------)

RomBug: ob 10
MPU type = 603, Input radix = 10
Ram (hard) breakpoints
Show control regs ON, Show FP regs ON in hex
Show MMU regs ON

RomBug: e
2  8 Machine Check stop on supervisor state
3  C Data Access stop on supervisor state
4  10 Instruction Access stop on supervisor state
6  18 Alignment stop on supervisor state
7  1C Program stop on supervisor state

RomBug: e
3  C Protection Violation <not monitored>
4  10 <Unassigned/Reserved> <not monitored>
2  8 Machine Check <not monitored>
6  18 Alignment Error <not monitored>
7  1C Program <not monitored>

RomBug: ov
RomBug: ov 2
2  8 Machine Check stop on supervisor/user state

RomBug: ov- 2
2  8 Machine Check <not monitored>
RomBug: ov 2 4
  2  8 Machine Check           stop on supervisor/user state
  3  C Data Access             stop on supervisor/user state
  4 10 Instruction Access      stop on supervisor/user state
RomBug: ov- 4
  4 10 Instruction Access      <not monitored>

RomBug: ovsd 4
  4 10 Instruction Access      display on supervisor state

RomBug: ovd 5
  5 14 External Interrupt      display on user state

RomBug: ovd 6
  6 18 Alignment               display on supervisor/user state

RomBug: ov
  2  8 Machine Check           stop on supervisor/user state
  3  C Data Access             stop on supervisor/user state
  4 10 Instruction Access      display on supervisor state
  5 14 External Interrupt      display on user state
  6 18 Alignment               display on supervisor/user state

RomBug: ov- 2 6
  2  8 Machine Check           <not monitored>
  3  C Data Access             <not monitored>
  4 10 Instruction Access      <not monitored>
  5 14 External Interrupt      <not monitored>
  6 18 Alignment               <not monitored>
The following information is provided in this section for the ARM processors:

- Options

Commands
Levels
Operating States
Operating Modes
Supported Registers
Display Information
Changing Machine Registers
Instruction Disassembly
Setting and Displaying Debug Options

Supported processors are:

- Level 3
- Level 4
-o Options

-o options identified in Table 6-1. Options are supported by the ARM version of RomBug.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Display registers upon monitored exception</td>
</tr>
<tr>
<td>a</td>
<td>Toggle control register display</td>
</tr>
<tr>
<td>d</td>
<td>Toggle FP decimal register display</td>
</tr>
<tr>
<td>f</td>
<td>Toggle FP register display</td>
</tr>
<tr>
<td>m</td>
<td>Toggle MMU register display</td>
</tr>
</tbody>
</table>

Commands

ARM-specific command information is provided in Table 6-2. Commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Enable/disable monitoring of processor-specific default exception vectors. Default vector numbers for exceptions are Undefined instruction= 1 Abort on instruction prefetch= 3 Abort on data access= 4 Alignment error= 8</td>
</tr>
</tbody>
</table>

Levels

Microware ARM Level 3 code uses svc mode (supervisor) for system state. Microware ARM Level 4 uses system mode for system state.

Level 4 supports full half word writes. Level 3 does not support half word access and actually writes as two bytes. Level 3 commands that reference half words cause two separate byte accesses.

Operating States

Two operating states are supported, ARM and THUMB. ARM state executes 32-bit, word-aligned ARM instructions. THUMB state operates with 16-bit, halfword-aligned THUMB instructions where the PC uses bit one to select between alternate halfwords.

Transitioning between states does not affect processor modes or register content but may affect the availability of particular registers.
RomBug runs in ARM mode and the default disassembly mode is ARM mode. The `ot` command toggles display of THUMB/ARM disassembly (not implemented).

**Operating Modes**

Six operating modes are supported. Modes are described in Table 6-3. Operating Modes.

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Normal user program execution state</td>
</tr>
<tr>
<td>FIQ</td>
<td>Supports data transfer or channel process</td>
</tr>
<tr>
<td>IRQ</td>
<td>General purpose interrupt handling</td>
</tr>
<tr>
<td>Supervisor</td>
<td>Protected mode for the operating system*</td>
</tr>
<tr>
<td>Abort</td>
<td>Entered after a data or instruction prefetch abort</td>
</tr>
<tr>
<td>System</td>
<td>Privileged user mode for the operating system #</td>
</tr>
<tr>
<td>Undefined</td>
<td>Entered when an undefined instruction is executed</td>
</tr>
</tbody>
</table>

†† = Level 4 only  
* Level 3 uses Supervisor for system state  
# Level 4 uses System for system state

**Supported Registers**

ARM architecture provides 15 general purpose, 32-bit registers, one 32-bit program counter register, ten floating point registers, and eight control registers. Availability of a particular register depends on the active processor state and operating mode.

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>Register Name</td>
<td>Alias Register Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>r2</td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r3</td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r4</td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r5</td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r6</td>
<td>gp</td>
<td>Global data pointer</td>
</tr>
<tr>
<td>r7</td>
<td></td>
<td>Integer/pointer function return value. For functions returning aggregates, this points to the returned aggregate, first required registrable function argument. Caller saved register for locals and temporaries</td>
</tr>
<tr>
<td>r8</td>
<td></td>
<td>Second required registrable argument. Caller saved register for locals and temporaries</td>
</tr>
<tr>
<td>r9</td>
<td></td>
<td>Third required registrable argument. Caller saved register for locals and temporaries</td>
</tr>
<tr>
<td>r10</td>
<td></td>
<td>Fourth required registrable argument. Caller saved register for locals and temporaries</td>
</tr>
<tr>
<td>r11</td>
<td></td>
<td>Caller saved register for locals and temporaries</td>
</tr>
<tr>
<td>r12</td>
<td>cp</td>
<td>Code constant pointer</td>
</tr>
<tr>
<td>r13</td>
<td>sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>r14</td>
<td>lr</td>
<td>Link register</td>
</tr>
<tr>
<td>r15</td>
<td>pc</td>
<td>Program counter</td>
</tr>
</tbody>
</table>
The ARM FPU holds all floating points in extended (three word) format internally. RomBug always displays these registers in the same format, extended.

Table 6-5. Floating Point (fp) Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td></td>
<td>First required ( f_p ) argument or the ( f_p ) function return value (caller-saved)</td>
</tr>
<tr>
<td>f1</td>
<td></td>
<td>Second required ( f_p ) argument (caller-saved)</td>
</tr>
<tr>
<td>f2</td>
<td></td>
<td>Third required ( f_p ) argument (caller-saved)</td>
</tr>
<tr>
<td>f3</td>
<td></td>
<td>Fourth required ( f_p ) argument (caller-saved)</td>
</tr>
<tr>
<td>f4</td>
<td></td>
<td>Callee-saved register</td>
</tr>
<tr>
<td>f5</td>
<td></td>
<td>Callee-saved register</td>
</tr>
<tr>
<td>f6</td>
<td></td>
<td>Callee-saved register</td>
</tr>
<tr>
<td>f7</td>
<td></td>
<td>Callee-saved register</td>
</tr>
<tr>
<td>fpcr0</td>
<td>fpsr</td>
<td>Floating point status register</td>
</tr>
<tr>
<td>fpcr15</td>
<td>fpcr*</td>
<td>Floating point control register</td>
</tr>
</tbody>
</table>

* Implementation dependent

The control register (CP15) read/write functions are defined in the ARM Architecture Reference document, System Control Coprocessor tables.

Table 6-6. Control Registers (CP15)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scc0</td>
<td>id</td>
<td>Processor ID</td>
</tr>
<tr>
<td>scc1</td>
<td>ctrl</td>
<td>Control register</td>
</tr>
<tr>
<td>scc2</td>
<td>ttbr</td>
<td>Translation table base register</td>
</tr>
</tbody>
</table>
ARM State

In all operating modes except system or user modes, some registers are banked. Banked registers are identified by an underscore in the register name, and by shading of the cells in Table 6-7. ARM State General Purpose Registers and Table 6-8. Current Program Status Registers (cpsr).

Table 6-6. Control Registers (CP15)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scc3</td>
<td>dacr</td>
<td>Domain access control register</td>
</tr>
<tr>
<td>scc4*</td>
<td>fsr</td>
<td>Fault status register</td>
</tr>
<tr>
<td>scc5</td>
<td>far</td>
<td>Fault address register</td>
</tr>
<tr>
<td>scc7</td>
<td>cf</td>
<td>Cache operations</td>
</tr>
<tr>
<td>scc8*</td>
<td>tlbf</td>
<td>Translation lookaside buffer (TLB) operations</td>
</tr>
<tr>
<td>scc9 - scc15</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

* Level 4 only

Table 6-7. ARM State General Purpose Registers

<table>
<thead>
<tr>
<th>System = &amp; User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
</tr>
<tr>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
</tr>
<tr>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
</tr>
<tr>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
</tr>
<tr>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
</tr>
</tbody>
</table>
THUMB State

Level 3 does not support THUMB State.

Registers available at one time in THUMB state are eight general purpose registers, one program counter register, a stack pointer (SP), a link register (LR), and the CPSR. In all operating modes except system/user mode, some registers are banked. Banked
registers are identified by an underscore in the register name, and by shading of the cells in Table 6-9, and Table 6-10.

Table 6-9. THUMB State General Purpose Registers

<table>
<thead>
<tr>
<th>System &amp; User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
</tr>
<tr>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
</tr>
<tr>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
</tr>
<tr>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
</tr>
<tr>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
<td>r6 (gp)</td>
</tr>
<tr>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
<td>r7</td>
</tr>
<tr>
<td>r13 (sp)</td>
<td>sp_fiq (sp)</td>
<td>sp_svc (sp)</td>
<td>sp_abt (sp)</td>
<td>sp_irq (sp)</td>
<td>sp_und (sp)</td>
</tr>
<tr>
<td>r14 (lr)</td>
<td>lr_fiq (lr)</td>
<td>lr_svc (lr)</td>
<td>lr_abt (lr)</td>
<td>lr_irq (lr)</td>
<td>lr_und (lr)</td>
</tr>
<tr>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
<td>r15 (pc)</td>
</tr>
</tbody>
</table>

Table 6-10. Current Program Status Registers (cpsr)

<table>
<thead>
<tr>
<th>System &amp; User</th>
<th>FIQ</th>
<th>Supervisor</th>
<th>Abort</th>
<th>IRQ</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpsr</td>
<td>cpsr</td>
<td>cpsr</td>
<td>cpsr</td>
<td>cpsr</td>
<td>cpsr</td>
</tr>
<tr>
<td>spsr_fiq</td>
<td>spsr_svc</td>
<td>spsr_abt</td>
<td>spsr_irq</td>
<td>spsr_und</td>
<td></td>
</tr>
</tbody>
</table>

Display Information

Register displays defined in the section are:
• Normal
• Program Status Register
• Floating Point Status Register

Normal Register

Normal register display:
00000000

Program Status Register

The Program Status Register (PSR) value is interpreted as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>User</td>
</tr>
<tr>
<td>10001</td>
<td>FIQ</td>
</tr>
<tr>
<td>10010</td>
<td>IRQ</td>
</tr>
<tr>
<td>10011</td>
<td>Supervisor</td>
</tr>
<tr>
<td>10111</td>
<td>Abort</td>
</tr>
</tbody>
</table>
Floating Point Status Register

The Floating Point Status Register (FPSR) value is interpreted as follows:

```
fpsr: 40001000 (--------------------------------)
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011</td>
<td>Undefined</td>
</tr>
<tr>
<td>11111</td>
<td>System =</td>
</tr>
</tbody>
</table>

= Level 4 only

Changing Machine Registers

Examples of changing and displaying machine registers follow:

RomBug: .
```
r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00000000
r5 : 00008CD0 r6 : 00100000 r7 : 00100950 r8 : 00100950 r9 : 00000000
r10: 00100210 r11: 00113FE0 r12: 00000000 sp : 00113FC4 lr : 00008DE4
pc: 00008DE4 cpsr: 20000013 (--C------------------------10011)
```
```
0x00008DE4   >288090E5         ldr r8,[r0,#0x28]
```

RomBug: .r4 100
```
r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00000010
r5 : 00008CD0 r6 : 00100000 r7 : 00100950 r8 : 00100950 r9 : 00000000
r10: 00100210 r11: 00113FE0 r12: 00000000 sp : 00113FC4 lr : 00008DE4
pc: 00008DE4 cpsr: 20000013 (--C------------------------10011)
```
```
0x00008DE4   >288090E5         ldr r8,[r0,#0x28]
```

RomBug: .r4 .r2+.r6
```
r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00200480
r5 : 00008CD0 r6 : 00100000 r7 : 00100950 r8 : 00100950 r9 : 00000000
r10: 00100210 r11: 00113FE0 r12: 00000000 sp : 00113FC4 lr : 00008DE4
pc: 00008DE4 cpsr: 20000013 (--C------------------------10011)
```
Instruction Disassembly

RomBug: `di Main`

```
main >0DB0AE1    mov r11,sp
main+0x4 >01502DE9  stmdb sp!, {r0,r12,lr}
main+0x8 >04D04DE2   sub sp,sp,#0x4
main+0xC >00B08DE5    str r11,[sp]
main+0x10 >000000EB   bl main+0x18-->
main+0x14 >C0CCFFFF   swi 0xFFCCC0
main+0x18 >00C09EE5   ldr r12,[lr]
main+0x1C >0EC08CE0   add r12,r12,lr
main+0x20 >0700A0E1    mov r0,r7
main+0x24 >407D86E2    add r7,r6,#0x1000
main+0x28 >CC7A97E5   ldr r7,[r7,#0xACC]
main+0x2C >000057E3    cmp r7,#0x0
main+0x30 >0400000A   beq main+0x48-->
main+0x34 >0070A0E3    mov r7,#0x0
main+0x38 >6C7D87E2    add r7,r7,#0x1B00
main+0x3C >067087E0    add r7,r7,r6
```

Setting and Displaying Debug Options

The `o` command displays and changes debugger modes. To display available options, use `o?`. The following examples show the use of the `o` command with each of its options:

RomBug: `o`

```
MPU type = 3, Input radix = 16
Ram (hard) breakpoints
Show control regs OFF
Show MMU regs OFF
```

RomBug: `o?`

```
RomBug Options:

b<n>              Numeric input base radix
s                  Toggle showing cpu registers during trace
v                  Display vectors being monitored
V[-][s][u][d]<n>  [<m>]  Monitor exception vector {'-' to restore vector}
        's' system state only, 'u' user state only
        'd' display only, <m> range of vectors
v?               Display all exception vector values

ARM Options:
```
.  Display registers upon monitored exception
a  Toggle control registers
d  Toggle FP decimal register display
f  Toggle FP register display
m  Toggle MMU register display

r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00000000
r5 : 00008CD0 r6 : 00100000 r7 : 00100950 r8 : 00100950 r9 : 00000000
r10: 00100210 r11: 00113FE0 r12: 00000000 sp : 00113FC4 lr : 0008DE4
pc: 00008DE4 cpsr: 20000093 (--C---------------------I--10011)
0x00008DE4  >288090E5         ldr r8,[r0,#0x28]

MPU type = 3, Input radix = 16
Ram (hard) breakpoints
Show control regs ON
Show MMU regs OFF

Processor id : 00000000 ctrl : 00000000 (----------)

r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00000000
r5 : 00008CD0 r6 : 00100000 r7 : 00100950 r8 : 00100950 r9 : 00000000
r10: 00100210 r11: 00113FE0 r12: 00000000 sp : 00113FC4 lr : 0008DE4
pc: 00008DE4 cpsr: 20000093 (--C---------------------I--10011)
0x00008DE4  >288090E5         ldr r8,[r0,#0x28]

MPU type = 3, Input radix = 16
Ram (hard) breakpoints
Show control regs ON
Show MMU regs ON

Processor id : 00000000 ctrl : 00000000 (----------)
ttbr 00000000  dacr 00000000   fsr 00000000   far 00000000
r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00000000
r5 : 00008CD0 r6 : 00100000 r7 : 00100950 r8 : 00100950 r9 : 00000000
r10: 00100210 r11: 00113FE0 r12: 00000000 sp : 00113FC4 lr : 0008DE4
pc: 00008DE4 cpsr: 20000093 (--C---------------------I--10011)
0x00008DE4  >288090E5         ldr r8,[r0,#0x28]
RomBug: **ob 10**

MPU type = 3, Input radix = 16

Ram (hard) breakpoints
Show control regs ON
Show MMU regs ON

RomBug: **e**

1. Undefined Instruction          stop on supervisor state
3. Abort on Instruction Prefetch  stop on supervisor state
4. Abort on Data Access           stop on supervisor state
8. Alignment error               stop on supervisor state

RomBug: **e**

1. Undefined Instruction          <not monitored>
3. Abort on Instruction Prefetch  <not monitored>
4. Abort on Data Access           <not monitored>
8. Alignment error               <not monitored>

RomBug: **ov**

RomBug: **ov 3**

3. Abort on Instruction Prefetch  stop on supervisor/user state

RomBug: **ov- 3**

3. Abort on Instruction Prefetch  <not monitored>

RomBug: **ov 3 5**

3. Abort on Instruction Prefetch  stop on supervisor/user state
4. Abort on Data Access           stop on supervisor/user state
5. <User defined/Reserved>       stop on supervisor/user state

RomBug: **ov- 4**

4. Abort on Data Access           <not monitored>

RomBug: **ovsd 4**

4. Abort on Data Access           display on supervisor state

RomBug: **ovud 5**

5. <User defined/Reserved>       display on supervisor/user state

RomBug: **ovd 6**

6. External Interrupt           display on supervisor/user state
RomBug: ov
3  Abort on Instruction Prefetch    stop on supervisor/user state
4  Abort on Data Access            display on supervisor state
5  <User defined/Reserved>         display on supervisor/user state
6  External Interrupt             display on supervisor/user state

RomBug: ov 3 6
3  Abort on Instruction Prefetch    <not monitored>
4  Abort on Data Access            <not monitored>
5  <User defined/Reserved>         <not monitored>
6  External Interrupt             <not monitored>
The following is provided in this section for SuperH processors:

- Options
- Commands
- Supported Registers
- Display Information
- Change Machine Registers
- Instruction Disassembly
- Watch Points

Supported processors are:

- SH7709
- SH7709A
- SH7750
- SH7780
-o Options

- o options identified in the following tables are supported by the SuperH version of RomBug.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b&lt;n&gt;</td>
<td>Numeric input base radix</td>
</tr>
<tr>
<td>r</td>
<td>Use rom type (soft) breakpoints</td>
</tr>
<tr>
<td>s</td>
<td>Toggle showing cpu registers during trace</td>
</tr>
<tr>
<td>v</td>
<td>Display vectors being monitored</td>
</tr>
<tr>
<td>v[-] [s</td>
<td>u][d]&lt;n&gt;[&lt;m&gt;]</td>
</tr>
<tr>
<td>v?</td>
<td>Display all exception vector values</td>
</tr>
<tr>
<td>x</td>
<td>Toggle disassembly hex output format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Display registers upon monitored exception</td>
</tr>
<tr>
<td>a</td>
<td>Toggle control registers display</td>
</tr>
<tr>
<td>d</td>
<td>Toggle floating-point decimal display</td>
</tr>
<tr>
<td>f</td>
<td>Toggle floating-point register display</td>
</tr>
<tr>
<td>k [&lt;watch_num&gt;]</td>
<td>Kill watch point(s)</td>
</tr>
<tr>
<td></td>
<td>&lt;watch_num&gt;: watch point number</td>
</tr>
<tr>
<td></td>
<td>If no &lt;watch_num&gt; is specified, kill all watch points</td>
</tr>
<tr>
<td>m</td>
<td>Toggle MMU registers display</td>
</tr>
<tr>
<td>w</td>
<td>Display table of active watch points/sequences</td>
</tr>
</tbody>
</table>
Chapter 7: SuperH Processors

Table 7-2. SH Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{w&lt;cond&gt;[#&lt;cond&gt; ...]}</td>
<td>Set a watch point or sequence per \texttt{&lt;cond&gt;}(s) (maximum number of \texttt{&lt;cond&gt;}s is eight); execution breaks when all \texttt{&lt;cond&gt;}s met in left-to-right order</td>
</tr>
<tr>
<td>\texttt{w?}</td>
<td>Above watch \texttt{&lt;cond&gt;} description and syntax help</td>
</tr>
</tbody>
</table>

Commands

SuperH-specific command information is provided in the following table.

Table 7-3. SuperH Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{e}</td>
<td>Enable/disable monitoring of processor-specific default exception vectors. Default vector numbers for exceptions are: Address error, load= 7 Address error, store= 8 Reserved instruction= C Illegal slot instruction= D</td>
</tr>
</tbody>
</table>

Supported Registers

SuperH architecture provides 16 32-bit general purpose registers, three control registers, and four system registers. Eight of the 16 general purpose registers are banked.

General Purpose Registers

Table 7-4. SuperH General Purpose Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias</th>
<th>Banked</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>r0b</td>
<td></td>
<td>Integral return values. Caller saved register for locals and temporaries.</td>
</tr>
<tr>
<td>r1</td>
<td>r1b</td>
<td></td>
<td>Caller saved register for locals and temporaries.</td>
</tr>
<tr>
<td>r2</td>
<td>r2b</td>
<td></td>
<td>Caller saved register for locals and temporaries.</td>
</tr>
<tr>
<td>r3</td>
<td>r3b</td>
<td></td>
<td>Caller saved register for locals and temporaries.</td>
</tr>
</tbody>
</table>
Using RomBug

Table 7-4. SuperH General Purpose Registers (Continued)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Alias</th>
<th>Banked</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r4</td>
<td>r4b</td>
<td></td>
<td>First function argument. Most significant half of double return value (SH-3 processors only). Caller saved register for locals and temporaries.</td>
</tr>
<tr>
<td>r5</td>
<td>r5b</td>
<td></td>
<td>Second function argument. Least significant half of double return value (SH-3 processors only). Caller saved register for locals and temporaries.</td>
</tr>
<tr>
<td>r6</td>
<td>r6b</td>
<td></td>
<td>Third function argument. Caller saved register for locals and temporaries.</td>
</tr>
<tr>
<td>r7</td>
<td>r7b</td>
<td></td>
<td>Fourth function argument.</td>
</tr>
<tr>
<td>r8</td>
<td></td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r9</td>
<td></td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r10</td>
<td></td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r11</td>
<td></td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r12</td>
<td></td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r13</td>
<td></td>
<td></td>
<td>Callee saved register for locals and temporaries</td>
</tr>
<tr>
<td>r14</td>
<td>gp</td>
<td></td>
<td>Global data pointer</td>
</tr>
<tr>
<td>r15</td>
<td>sp</td>
<td></td>
<td>Stack pointer</td>
</tr>
</tbody>
</table>

Control Registers

Table 7-5. SuperH Control Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sr</td>
<td>Status register</td>
</tr>
<tr>
<td>gbr</td>
<td>Global base register</td>
</tr>
<tr>
<td>vbr</td>
<td>Vector base register</td>
</tr>
</tbody>
</table>
System Registers

Table 7-6. SuperH System Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mach</td>
<td>Multiply and accumulate high register</td>
</tr>
<tr>
<td>macl</td>
<td>Multiply and accumulate low register</td>
</tr>
<tr>
<td>pr</td>
<td>Procedure register</td>
</tr>
<tr>
<td>pc</td>
<td>Program counter</td>
</tr>
</tbody>
</table>

Floating Point Registers (SH-4 and SH-4A only)

Table 7-7. SH-4 Floating Point Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Banked</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td>x0</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f1</td>
<td>x1</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f2</td>
<td>x2</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f3</td>
<td>x3</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f4</td>
<td>x4</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f5</td>
<td>x5</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f6</td>
<td>x6</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f7</td>
<td>x7</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f8</td>
<td>x8</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f9</td>
<td>x9</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f10</td>
<td>x10</td>
<td>Floating point single precision register</td>
</tr>
<tr>
<td>f11</td>
<td>x11</td>
<td>Floating point single precision register</td>
</tr>
</tbody>
</table>
Display Information

Register displays defined in this section are:

- Normal
- Status
- MMU Control
- Floating Point Status and Control

Normal Register Display

The normal register display for SuperH is:

```
00000000
```
Status Register Display

- FD (FPU disable bit): cleared to 0 by reset
- I3-I0 bits: interrupt mask bits
- sr: 400001F2 (-S0e--mQ1111--Sf)
- T bit: f = false
- S bit: S = set
- Q bit: Q = set
- M bit: m = clear
- BL bit: e = exceptions enabled
- RB bit: 0 = bank 0 registers
- MD bit: S = system or privileged mode

* The FD bit is defined only for the SH-4

MMU Control Register Display

Default value is 00000000.

- mmucr: 00000000 (m--00-fxd)
- AT (address translation bit): d = disabled
- IX (index mode bit): See the Hitachi SuperH Hardware Manual for more information.
- TF (TLB flush bit): always reads 0
- RC: two bit random counter. See the Hitachi SuperH Hardware Manual for more information.
**MMU Control Register Display (SH-4/SH-4A)**

Default value is 00000000.

\[ \text{mmucr: 1C--88-- (m--00-fxd)} \]

- **AT (address translation bit):**
  - \( d \)=disabled
  - Reserved bit.

- **SV (single virtual memory mode bit):**
  - \( m \)=multiple virtual memory mode

- **TI (TLB invalidate bit):** always reads 0.
  - Reserved: default = 0

---

**Floating Point Status and Control Register Display (SH-4/SH-4A)**

\[ \text{fpscr:00000000 (----------0000-----------------00)} \]

- **PR (Precision Mode Bit):**
  - \( 0 \)=Floating-point instructions are executed as single-precision instructions

- **DN (Denormalization Mode Bit):**
  - \( 0 \)=a denormalized number is treated as a denormalized number

- **SZ (Transfer Size Mode Bit):** \( 0 \)=FMOV transfers a single-precision floating-point number

- **RM bits:**
  - \( 00 \)=round to nearest number

- **FR bit:** Floating-point register bank
Change Machine Registers

Example

RomBug: .
r0: 00000788 8C0012D0 8C0012D0 9FFFF8CB2 8C000E3C 8C000000 8C000E3C 800292B0
r8: 8C000E3C 8C000DF8 8C000EBC 00000001 00000000 E0007FFC 8C007FFC 8C023FC4
pr: 80000CE6 mach: FFFFFFFF macl: FFFFFPPP gbr: FFFFFPPP
pc: 80000CE6 sr: 400001F2 (-S0e------------------mQ1111--Sf)
0x80000CE6 >518A    mov.l @(0x28,r8),r1
RomBug: .r4 100
RomBug: .
r0: 00000788 8C0012D0 8C0012D0 9FFFF8CB2 00000100 8C000000 8C000E3C 800292B0
r8: 8C000E3C 8C000DF8 8C000EBC 00000001 00000000 E0007FFC 8C007FFC 8C023FC4
pr: 80000CE6 mach: FFFFFPPP macl: FFFFFPPP gbr: FFFFFPPP
pc: 80000CE6 sr: 400001F2 (-S0e------------------mQ1111--Sf)
0x80000CE6 >518A    mov.l @(0x28,r8),r1
RomBug: .r4 ..r2+.r6
RomBug: .
r0: 00000788 8C0012D0 8C0012D0 9FFFF8CB2 1800210C 8C000000 8C000E3C 800292B0
r8: 8C000E3C 8C000DF8 8C000EBC 00000001 00000000 E0007FFC 8C007FFC 8C023FC4
pr: 80000CE6 mach: FFFFFPPP macl: FFFFFPPP gbr: FFFFFPPP
pc: 80000CE6 sr: 400001F2 (-S0e------------------mQ1111--Sf)
0x80000CE6 >518A    mov.l @(0x28,r8),r1
RomBug:

RomBug: .f0 4
RomBug: .
f0: 4 0 0 0
f4: 0 0 0 0
f8: 0 0 0 0
f12: 0 0 0 0
x0: 0 0 0 0
x4: 0 0 0 0
x8: 0 0 0 0
x12: 0 0 0 0
dr0: 512 0 0 0
dr8: 0 0 0 0
xd0: 0 0 0 0
xd8: 0 0 0 0
fpscr: 00000000 (----------0000-----------------00)
r0: 00000A04 8C15ECB8 8C001360 00000000 00000000 00000000 8FFDFBEC 00000000
r8: 40000001 00000000 0C15EDD0 00000000 00000000 00000000 8C008980 8FDFDF80
pr: 8C02E874 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C02E874 sr: 400000F1 (-S0e------------------mQ1111--sT)
0x8C02E874 >480E    ldc r8,sr
RomBug: .f4 3.14159
RomBug: .
f0: 4 0 0 0
f4: 3.14159012 0 0 0
f8: 0 0 0 0
f12: 0 0 0 0
x0: 0 0 0 0
x4: 0 0 0 0
x8: 0 0 0 0
x12: 0 0 0 0
dr0: 512 0 50.1235352 0
dr8: 0 0 0 0
xd0: 0 0 0 0
xd8: 0 0 0 0
RomBug: `dx &.f4`

MPU type = 4, Input radix = 16
Ram (hard) breakpoints
Show control regs OFF, Show FP regs ON in hex
Show MMU regs OFF
RomBug:
Watch Points

The SuperH processor has two built in watch point registers. Rombug can fully utilize these registers using the “ow” command. Instruction type watch points can be set, as well as data type watch points. Data type watch points can be further defined as to a read and/or write type access, and 8 bit, 16 bit, or 32 bit access.

There is one caveat to using watch points in user state. Since on the SuperH processor, system state runs in P1 page, and user state runs in U0 page, there is a difference in addresses between system and user state (ie. system state addresses have the high bit set). Rombug runs in system state and understands system state addresses. It will not translate any addresses to user state. Since the watch point registers are looking for an exact address, the address set to the watch point must be properly translated depending if the code to be debugged will run in user state or system state.

Watch Point Example 1

This first example sets a instruction watch point in system state.

```
[32]$ break
<Called>
r0: 000000A0 8C16FEC8 8C001360 00000000 00000000 8FFC1A0C 00000000
r8: 40008001 00000000 00000000 00000000 00000000 00000000 8C008A40 8FFC19D0
pr: 8C02E6C4 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C02E6C4 sr: 400080F1 (-S0e------------------mq1111--sT)
0x8C02E6C4  >480E   ldc r8,sr
RomBug: a sc16550
RomBug: sc
text              C 8C048B70  btext              C 8C048B70
bname              C 8C048BC8  bname              C 8C048BC8
main               C 8C048BD0  init               C 8C048BD8
config             C 8C048DB8  irqsort            C 8C048F10
```
Using RomBug

input_irq   C 8C048FDC  output_irq     C 8C049286
status_irq  C 8C0493D0  entxirq       C 8C04944E
read        C 8C0494E0  getstat        C 8C0494EB
setstat     C 8C049502  terminate      C 8C049630
write       C 8C0496A0  __pic_enable   C 8C0496A8
__pic_disable C 8C0496AC __os_irq       C 8C0496B0
__oscall     C 8C0496E0  __os_send      C 8C0496F0
inw         C 8C049718  outw           C 8C049720
irq_disable C 8C049728  irq_maskget    C 8C049738
irq_enable  C 8C04974C  irq_save       C 8C049760
irq_restore C 8C04976E  irq_change     C 8C04977C
change_static C 8C04979A get_static     C 8C04979A
__etext     C 8C0498A0  etext           C 8C0498A0

RomBug: ow?

============ Condition <cond> Syntax of Watch (ow) Command ============
<cond> = a <addr>[,<addr_mask>]
   Condition matches on any access of any size to the memory at <addr>,
   including data read, data write and instruction pre-execution.
or <cond> = i <addr>[,<addr_mask>]
   Condition matches on execution of the instruction at <addr>
or <cond> = <mode>[.<size>] [<addr>[,<addr_mask>]][;<data_spec>]
   Condition matches on <size> size <mode> access of memory at <addr>,
   with the data value of <data_spec> if specified, otherwise with any
   value. Only one watch point / sequence may contain <data_spec>(s).
<addr>:  Trigger address specified as an <expression>
<addr_mask>: If specified, the <addr_mask> least significant bits of
           <addr> are ignored (XBITS). Available values: 10, 12, 16, 20
<mode>:  Access Mode, one of:
         d = Data Read/Write, r = Data Read, w = Data Write
<size>:  Access Size, one of:
         b = byte (8 bit), w = word (16 bit), l = long (32 bit),
           q = quadword (64 bit), (none) = any size
<data_spec>:  <data>[,<data_mask>]
<data>:  Trigger Data specified as an <expression>
<data_mask>:  Data Mask specified as an <expression>; bits which are
             set in <data_mask> are ignored (don't care) in <data>
<expression>: Meaningful combination of constants, labels, operators

RomBug: owi init
Watch Point # 0 set.
RomBug: ow

WP # ACCESS ADDRESS XBITS SIZE DATA MASK
===============================================
00     Instr 0xC048BD8 0  Word ----------
01     Instr 0xC048BD8 0  Word ----------
RomBug: g

***Warning*** - breakpoints halt timesharing.
[33]$ iniz t3
<at watchpoint>
ro: 00000234 8C048BD8 0000000C 8FFC2A4 8FFC8AE0 8FFC8AE0 8C0419DA 8FFEC2A0
r8: 00000000 8FFBC70 8FFC8AE0 8FFC2A0 8FFC8990 00000002 8FFC8990 8FFC18B4
pr: 8C041A50 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C048BD8 sr: 40080000 (-S0e------------------mq0000--sf)
init >2F96     mov.l r9,@-r15
RomBug: ow

WP # ACCESS ADDRESS XBITS SIZE DATA MASK
===============================================
00     Instr 0xC048BD8 0  Word ----------
01     Instr 0xC048BD8 0  Word ----------
RomBug: g
<at watchpoint>
ro: 00000234 8C048BD8 0000000C 8FFC2A4 8FFC8AE0 8FFC8AE0 8C0419DA 8FFEC2A0
Watch Point Example 2

The second example is an instruction break point from user state.

[b6]$ break
<Called>
[65]$ break
r0: 00000A04 8C16FEC8 8C001360 00000000 00000000 00000000 00000000 00000000
r8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
pr: 0FFB5D9A mach: 00000000 macl: 00000000 gbr: 00000000
pc: 0FFB5D9A sr: 00008001 (-u0e------------------mq0000--sT)
0x8C02E6C4  >480E             ldc r8,sr
RomBug: a f_strap
RomBug: .rr1 main
RomBug: .rr
RRn:00000000 8FFB5D9A 00000000 00000000 00000000 00000000 00000000 00000000
RomBug: di .rr1
main     >93AE             mov.w @(main+0x160,pc),r3 (#0xffffffff68)
main+0x2 >012A             sts pr,r1
main+0x4 >33FC             add r15,r3
main+0x6 >909C             mov.w @(main+0x142,pc),r0 (#0xffffffff8050)
main+0x8 >02EE             mov.l @(r0,r14),r2
main+0xA >3236             cmp.hi r3,r2
main+0xC >8B02             bf main+0x18
main+0xE >90A8             mov.w @(main+0x162,pc),r0 (#0x662)
main+0x10 >00030009         bsrf r0
main+0x14 >2F96             mov.l r9,@-r15
main+0x16 >2FB6             mov.l r8,@-r15
main+0x18 >6F33             mov r3,r15
main+0x1A >2F12             mov.l r1,r15
main+0x1C >E900             mov #0x00,r9
main+0x1E >E648             mov #0x48,r6
main+0x20 >7648             add #0x48,r6
dis: owi ffb5d9a
Watch Point # 0 set.
RomBug: ow

[b6]$ _f_strap
<at watchpoint>
r0: 0000048A 0C16C908 00002000 0C16EB48 00000001 0C16EB48 0C16EB50 0C16C908
r8: 0C16EB90 00000000 00000000 00000000 00000000 00000000 00000000 00000000
pr: 0FFB5910 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 0FFB5D9A sr: 00008001 (-u0e------------------mq0000--sT)
Watch Point Example 3

The third example is a watch point set on a long word write to the system global d_proc (offset 0x80).

```
[72]$ break
<Called>
<at watchpoint>
```

```
WP # ACCESS ADDRESS XBITS SIZE DATA MASK
00 Write 0x8C008AC0 0 Long ---------------
01 ------------------------ Not Set ------------------------
```

```
RomBug: g
***Warning*** - breakpoints halt timesharing.
<at watchpoint>
```

```
WP # ACCESS ADDRESS XBITS SIZE DATA MASK
00 Write 0x8C008AC0 0 Long ---------------
01 ------------------------ Not Set ------------------------
```

```
RomBug: g
```

```
WP # ACCESS ADDRESS XBITS SIZE DATA MASK
00 Write 0x8C008AC0 0 Long ---------------
01 ------------------------ Not Set ------------------------
```

```
RomBug: g
```

```
[73]$
```
MIPS Processors

The following is provided in this section for 79R4700 64-bit MIPS processors:

- o Options
  Commands
  Supported Registers
  Display Information
  Setting Breakpoints
  Trace Command

Supported processors are:

- IDT 79R4700
-o Options

- o options identified in the following tables are supported by the 79R4700 version of RomBug.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b&lt;n&gt;</td>
<td>Numeric input base radix</td>
</tr>
<tr>
<td>r</td>
<td>Use ROM type (soft) breakpoints (not supported by the 79R4700)</td>
</tr>
<tr>
<td>s</td>
<td>Toggle showing cpu registers during trace</td>
</tr>
<tr>
<td>v</td>
<td>Display vectors being monitored</td>
</tr>
<tr>
<td>v[-] [s</td>
<td>u] [d] &lt;n&gt;[&lt;m&gt;]</td>
</tr>
<tr>
<td>v?</td>
<td>Display all exception vector values</td>
</tr>
<tr>
<td>x</td>
<td>Toggle disassembly hex output format</td>
</tr>
</tbody>
</table>
Table 8-2. 79R4700 MIPS Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display registers upon monitored exception</td>
</tr>
<tr>
<td>a</td>
<td>Toggle control registers display</td>
</tr>
<tr>
<td>d</td>
<td>Toggle FP decimal register display</td>
</tr>
<tr>
<td>f</td>
<td>Toggle FP register display</td>
</tr>
<tr>
<td>k [&lt;watch_num&gt;]</td>
<td>Kill watch point(s) (not supported by the 79R4700)</td>
</tr>
<tr>
<td>m</td>
<td>Toggle MMU registers display</td>
</tr>
<tr>
<td>w</td>
<td>Display table of active watch points/sequences</td>
</tr>
<tr>
<td>w&lt;cond&gt;[#&lt;cond&gt; ...]</td>
<td>Set a watch point or sequence per &lt;cond&gt; (s) (not supported by the 79R4700) (maximum number of &lt;cond&gt;s is eight); execution breaks when all &lt;cond&gt;s met in left-to-right order</td>
</tr>
<tr>
<td>w?</td>
<td>Above watch &lt;cond&gt; description &amp; syntax help (not supported by the 79R4700)</td>
</tr>
</tbody>
</table>

Commands

79R4700-specific command information is provided in the following table.

Table 8-3. 79R4700 Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Enable/disable monitoring of processor-specific default exception vectors. Default vector numbers for exceptions are: TLB load= 2, TLB store= 3, Address error (load)= 4, Address error (write)= 5, Bus Error Exception (Fetch)= 6, Bus Error Exception (load/store)= 7, Co-Processor Unusable Exception= B, Arithmetic Overflow Exception= C</td>
</tr>
</tbody>
</table>
Supported Registers

The 79R4700 architecture provides 32 64-bit general purpose registers, one 64-bit program counter register, two 64-bit multiply and divide registers, 32 64-bit floating point coprocessor registers, and 32 64-bit system control coprocessor registers.

General Purpose Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>RomBug Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>zero</td>
<td>Constant Zero.</td>
</tr>
<tr>
<td>r1</td>
<td>at</td>
<td>Assembler temporary storage.</td>
</tr>
<tr>
<td>r2-r3</td>
<td>v0-v1</td>
<td>Function return.</td>
</tr>
<tr>
<td>r4-r7</td>
<td>a0-a3</td>
<td>Incoming args.</td>
</tr>
<tr>
<td>r8-r15</td>
<td>t0-t7</td>
<td>Registers for temporaries</td>
</tr>
<tr>
<td>r16-r23</td>
<td>s0-s7</td>
<td>Saved temporaries</td>
</tr>
<tr>
<td>r24-r25</td>
<td>t8-t9</td>
<td>Registers for temporaries</td>
</tr>
<tr>
<td>r26-27</td>
<td>k0-k1</td>
<td>Exception handling</td>
</tr>
<tr>
<td>r28</td>
<td>gp</td>
<td>Global data pointer</td>
</tr>
<tr>
<td>r29</td>
<td>sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>r30</td>
<td>cp</td>
<td>Saved temporary</td>
</tr>
<tr>
<td>r31</td>
<td>ra</td>
<td>Return Address</td>
</tr>
</tbody>
</table>
Multiply and Divide Registers

Table 8-5. 79R4700 Multiply and Divide Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>RomBug Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>hi</td>
<td>Multiply and Divide register Higher result</td>
</tr>
<tr>
<td>LO</td>
<td>lo</td>
<td>Multiply and Divide register Lower result</td>
</tr>
</tbody>
</table>

Program Counter Register

Table 8-6. 79R4700 Program Counter Register

<table>
<thead>
<tr>
<th>Register Name</th>
<th>RomBug Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>pc</td>
<td>Program Counter register</td>
</tr>
</tbody>
</table>

System Control Registers

There are 32 64-bit registers associated with the system control coprocessor.

Table 8-7. 79R4700 System Control Registers

<table>
<thead>
<tr>
<th>Register Number</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Index</td>
<td>Programmable Pointer into TLB array</td>
</tr>
<tr>
<td>1</td>
<td>Random</td>
<td>Pseudorandom Pointer into TLB array (read only)</td>
</tr>
<tr>
<td>2</td>
<td>EntryLo0</td>
<td>Low half of TLB entry for even virtual page (VPN)</td>
</tr>
<tr>
<td>3</td>
<td>EntryLo1</td>
<td>Low half of TLB entry for odd virtual page (VPN)</td>
</tr>
<tr>
<td>4</td>
<td>context</td>
<td>Pointer to kernel virtual page table entry (PTE) for 32-bit address spaces</td>
</tr>
<tr>
<td>5</td>
<td>PageMask</td>
<td>TLB Page Mask</td>
</tr>
<tr>
<td>6</td>
<td>Wired</td>
<td>Number of wired TLB entries</td>
</tr>
<tr>
<td>7</td>
<td>---</td>
<td>Reserved</td>
</tr>
<tr>
<td>Register Number</td>
<td>Register Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>8</td>
<td>BadVaddr</td>
<td>Bad virtual address</td>
</tr>
<tr>
<td>9</td>
<td>Count</td>
<td>Timer Count</td>
</tr>
<tr>
<td>10</td>
<td>EntryHi</td>
<td>High half of TLB entry</td>
</tr>
<tr>
<td>11</td>
<td>Compare</td>
<td>Timer Compare</td>
</tr>
<tr>
<td>12</td>
<td>SR</td>
<td>Status Register</td>
</tr>
<tr>
<td>13</td>
<td>Cause</td>
<td>Cause of last exception</td>
</tr>
<tr>
<td>14</td>
<td>EPC</td>
<td>Exception Program Counter</td>
</tr>
<tr>
<td>15</td>
<td>PRIId</td>
<td>Processor Revision Identifier</td>
</tr>
<tr>
<td>16</td>
<td>Config</td>
<td>Configuration register</td>
</tr>
<tr>
<td>17</td>
<td>LLAddr</td>
<td>Load Linked Address</td>
</tr>
<tr>
<td>18-19</td>
<td>---</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>XCText</td>
<td>Pointer to kernel virtual PTE table for 64-bit address spaces</td>
</tr>
<tr>
<td>21-25</td>
<td>---</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>ECC</td>
<td>Secondary-cache error checking and correcting (ECC) and Primary parity</td>
</tr>
<tr>
<td>27</td>
<td>CacheErr</td>
<td>Cache Error and Status register</td>
</tr>
<tr>
<td>28</td>
<td>TagLo</td>
<td>Cache Tag register</td>
</tr>
<tr>
<td>29</td>
<td>TagHi</td>
<td>Cache Tag register</td>
</tr>
<tr>
<td>30</td>
<td>ErrorEPC</td>
<td>Error Exception Program Counter</td>
</tr>
<tr>
<td>31</td>
<td>---</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Floating Point General Purpose Registers

The floating point general purpose registers (FGR) have two modes. When the FR bit (bit 26) in the processor’s status register (CCP0 #12) is set to 0, then the floating point registers are set up to be 16 64-bit registers for double-precision values or 32 32-bit registers for single precision values. When the FR bit (bit 26) in the processor’s status register (CCP0 #12) is set to 1, then the floating point registers are set up to be thirty-two 64-bit registers where each register can hold either single-precision or double-precision values. Regardless of the setting of the FR bit, RomBug will display all FPU registers as doubles.

<table>
<thead>
<tr>
<th>Floating-Point Register (FPR) Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>df0</td>
</tr>
<tr>
<td>df1</td>
</tr>
<tr>
<td>df2</td>
</tr>
<tr>
<td>df3</td>
</tr>
<tr>
<td>df28</td>
</tr>
<tr>
<td>df29</td>
</tr>
<tr>
<td>df30</td>
</tr>
<tr>
<td>df31</td>
</tr>
</tbody>
</table>

Display Information

Register displays defined in this section are:

- Normal
- Status
- Floating Point Status and Control

Normal Register Display

The normal register display for 79R4700 is:

```
0000000000000000
```
Using RomBug

Status Register Display

The status register (SR) is System Control Register number 12.

```
sr:400001F2 (---U-F--------DNIIINSSxxxKKNND)
   bit 31 bit 0
```

<table>
<thead>
<tr>
<th>Bit Field Number</th>
<th>Bit Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IE</td>
<td>Interrupt Enable. D = disabled, E = enabled.</td>
</tr>
<tr>
<td>1</td>
<td>EXL</td>
<td>Exception Level. N= normal, K = exception.</td>
</tr>
<tr>
<td>2</td>
<td>ERL</td>
<td>Error Level. N = normal, E = error.</td>
</tr>
<tr>
<td>3 - 4</td>
<td>KSU</td>
<td>Mode bits. UR = User, SU = Supervisor, KK = Kernel</td>
</tr>
<tr>
<td>5</td>
<td>UX</td>
<td>Enables 64-bit virtual addressing and operations in User mode. 0 = 32-bit, x = 64-bit</td>
</tr>
<tr>
<td>6</td>
<td>SX</td>
<td>Enables 64-bit virtual addressing and operations in Supervisor mode. 0 = 32-bit, x = 64-bit</td>
</tr>
<tr>
<td>7</td>
<td>KX</td>
<td>Determines if the TLB Refill Vector or the XTLB Refill Vector address is used for the TLB misses on kernel addresses. 0 = TLB Refill Vector, x = XTLB Refill Vector</td>
</tr>
<tr>
<td>8 - 15</td>
<td>IM</td>
<td>Interrupt Mask. Controls disabling and enabling interrupts.</td>
</tr>
<tr>
<td>16</td>
<td>DE</td>
<td>Specifies that cache parity errors cannot cause exceptions. E = enabled, D = Disabled.</td>
</tr>
<tr>
<td>17</td>
<td>CE</td>
<td>Must be 0.</td>
</tr>
<tr>
<td>18</td>
<td>CH</td>
<td>Read only.</td>
</tr>
<tr>
<td>19</td>
<td>O</td>
<td>Reserved. Always reads zeros. Must have zeros written to it.</td>
</tr>
<tr>
<td>20</td>
<td>SR</td>
<td>Read only.</td>
</tr>
</tbody>
</table>
Table 8-9. Status Register Bit Field Assignments

<table>
<thead>
<tr>
<th>Bit Field Number</th>
<th>Bit Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>0</td>
<td>Reserved. Always reads zeros. Must have zeros written to it.</td>
</tr>
<tr>
<td>22</td>
<td>BEV</td>
<td>Do not set. 0 indicates normal placement of vectors code</td>
</tr>
<tr>
<td>23 - 24</td>
<td>0</td>
<td>Reserved. Always reads zeros. Must have zeros written to it.</td>
</tr>
<tr>
<td>25</td>
<td>RE</td>
<td>Do not set. 0 indicates normal operation.</td>
</tr>
<tr>
<td>26</td>
<td>FR</td>
<td>Enable more floating point registers.</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>Reserved. Always reads zeros. Must have zeros written to it.</td>
</tr>
<tr>
<td>28 - 31</td>
<td>CU</td>
<td>Controls the usability of each of the four coprocessor unit numbers. U = usable</td>
</tr>
</tbody>
</table>
Floating Point Status and Control Register (FCR31)

Condition bit:
Floating point compare results are stored here.

Flag Bits:
Exception case detected, however not enabled.

I = Inexact operation
U = Underflow
O = Overflow
Z = Division by zero
V = Invalid operation

RM bits:
00 = round to nearest number
01 = round toward zero
10 = round toward positive infinity
11 = round toward

fcsr:00000000 (---------C-----EVZOUIVZOUIVZOU100)

Cause bits:
Written by each floating point operation. If set, can cause an exception.

E = Software Emulation is

Enable bits:
When set, corresponding bit causes appropriate exception.
### Rombug Examples

#### Setting Breakpoints

Setting breakpoints is done with the `b` command. An illustration of the command's usage follows. It sets a breakpoint at two labels: `dbg6` and `dbgR`.

```
[1]$ break <Called>
at:0000000000000000 v0:0000000000000020 v1:00000000000be060 a0:0000000000000000
a1:0000000000000000 a2:FFFFFFFF801caaf20 a3:0000000000000020 t0:FFFFFFFF801a1900
t1:FFFFFFFFfffffff e t2:0000000014017b00 t3:0000000000000001 t4:0000000000000000
t5:0000000000000001 t6:000000000001c6ff0 t7:000000000001c6ff0 a0:0000000014017b01
s1:0000000000000000 s2:0000000000000000 s3:0000000000000000 s4:0000000000000000
s5:0000000000000000 s6:0000000000000000 s7:0000000000000000 a3:0000000000000000
s8:0000000000000000 a6:0000000000000000 a7:0000000000000000 t8:FFFFFFFF803c6b0c
t9:FFFFFFFF801b3378 k0:00000000000be060 k1:FFFFFFFF801caea8 gp:FFFFFFFF801b0000
sp:FFFFFFFF803caea8 cp:0000000000000000 ra:FFFFFFFF80070eb0 hi:001C2F70001c2ec8
lo:0000000000000000 pc:FFFFFFFF80070e0b
sr:14017B00 (---U-F--------DNIIINESXXKKNNKE)
ox80070B0 >40960000 mtc0 a0,$12
RomBug: a spaconic
RomBug: b dbg6
RomBug: b dbgR
RomBug: g
***Warning*** - breakpoints halt timesharing.
[2]$ ipstart
[2]$ ipstart
[2]$ ipstart
<at breakpoint>
at:FFFFFFFF8007cd58 v0:0000000000000080 v1:00000000001171b0 a0:0000000014017b01
a1:00000000000001c4 a2:FFFFFFFFfffffff f a3:0000000000000000 t0:0000000000000000
a4:0000000000000000 t1:0000000000000000 t2:0000000000000000 t3:0000000000000000
sp:FFFFFFFF803caea8 cp:0000000000000000 ra:FFFFFFFF80144ede h1:0000000000000000
lo:0000000000000000 pc:FFFFFFFF80144e80
sr:14017B03 (---U-F--------DNIIINESXXKKNNKE)
dbg6 >8FA40010 lw a0,0x10(sp)
```

#### Trace Command

The following example illustrates the trace and register display commands.

```
RomBug: t
at:FFFFFFFF8007cd58 v0:0000000000000080 v1:00000000001171b0 a0:0000000014017b01
a1:00000000000001c4 a2:FFFFFFFFfffffff f a3:0000000000000000 t0:0000000000000000
a4:0000000000000000 t1:0000000000000000 t2:0000000000000000 t3:0000000000000000
sp:FFFFFFFF803caea8 cp:0000000000000000 ra:FFFFFFFF80144ede h1:0000000000000000
lo:0000000000000000 pc:FFFFFFFF80144e84
sr:14017B03 (---U-F--------DNIIINESXXKKNNKE)
dbg6+0x4 >8FA50008 lw a1,0x8(sp)
trace: d .a0
```
<table>
<thead>
<tr>
<th>Address</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
</table>
| 0x801CC5B8 | BP600000   | A0370000 A0370040 00000004 | 7.. 7.. 7.. | 0x801CC5B8
| A03700C0   | 00000001   | A03701C4 00000004 7.. 7.. 7.. | 0x801CC5B8 |
| 0x801CC58  | 002A0000   | 00000000 00000000 00000000 | 00000000 |
| 0x801CC68  | 0000000F   | 00000000 00000004 00000004 80381004 |
| 0x801CC68  | 6462675F   | 73707369 6E000000 00000000 |
| dbg_spsin  | 0x801CC68  | 0x801B0000 00000000 00000000 00000000 |
| 0x801CC68  | 0x801CC68  | 0x8013F90C 8013FD44 8013FE44 |
| 0x801CC68  | 8013F90C   | 8013FD44 8013FE44 |
| 0x801CC68  | 8013FA20   | 8013FA00 00000001 00000000 |
| 0x801CC68  | 80086944   | 00000000 00000000 00000000 |
| 0x801CC68  | 0x80102B1C | 0x8013FA00 00000001 00000000 |
| 0x801CC68  | 6462675F   | 73707369 6E000000 00000000 |
| 0x801CC68  | 0x801B0000 | 00000000 00000000 00000000 |
| 0x801CC68  | 0x8013F90C | 8013FD44 8013FE44 |
| 0x801CC68  | 8013FA20   | 8013FA00 00000001 00000000 |
| 0x801CC68  | 80086944   | 00000000 00000000 00000000 |
| 0x801CC68  | 0x80102B1C | 0x8013FA00 00000001 00000000 |
| 0x801CC68  | 6462675F   | 73707369 6E000000 00000000 |
| 0x801CC68  | 0x801B0000 | 00000000 00000000 00000000 |
| 0x801CC68  | 0x8013F90C | 8013FD44 8013FE44 |
| 0x801CC68  | 8013FA20   | 8013FA00 00000001 00000000 |
| 0x801CC68  | 80086944   | 00000000 00000000 00000000 |
| 0x801CC68  | 0x80102B1C | 0x8013FA00 00000001 00000000 |
| 0x801CC68  | 6462675F   | 73707369 6E000000 00000000 |
| 0x801CC68  | 0x801B0000 | 00000000 00000000 00000000 |
| 0x801CC68  | 0x8013F90C | 8013FD44 8013FE44 |
| 0x801CC68  | 8013FA20   | 8013FA00 00000001 00000000 |
| 0x801CC68  | 80086944   | 00000000 00000000 00000000 |
| 0x801CC68  | 0x80102B1C | 0x8013FA00 00000001 00000000 |
| 0x801CC68  | 6462675F   | 73707369 6E000000 00000000 |
| 0x801CC68  | 0x801B0000 | 00000000 00000000 00000000 |
| 0x801CC68  | 0x8013F90C | 8013FD44 8013FE44 |
| 0x801CC68  | 8013FA20   | 8013FA00 00000001 00000000 |
| 0x801CC68  | 80086944   | 00000000 00000000 00000000 |
| 0x801CC68  | 0x80102B1C | 0x8013FA00 00000001 00000000 |
The following is provided in this section for SH-5 family processors:

- **Options**
- **Commands**
- **Supported Registers**
- **Display Information**
- **Setting Breakpoints**
- **Trace Command**

The supported processors include SH8001.
-o Options

-o options identified in the following tables are supported by the SH-5 version of RomBug.

### Table 9-1. SH-5 RomBug Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b&lt;n&gt;</td>
<td>Numeric input base radix</td>
</tr>
<tr>
<td>r</td>
<td>Use ROM type (soft) breakpoints</td>
</tr>
<tr>
<td>s</td>
<td>Toggle showing cpu registers during trace</td>
</tr>
<tr>
<td>v</td>
<td>Display vectors being monitored</td>
</tr>
<tr>
<td>v [-] [s</td>
<td>u</td>
</tr>
<tr>
<td>v?</td>
<td>Display all exception vector values</td>
</tr>
</tbody>
</table>

- ‘-’ = disable monitoring
- ‘s’ = system state only
- ‘u’ = user state only
- ‘d’ = display only
- <m> = end of range of vectors
## Table 9-2. SH-5 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Display registers upon monitored exception</td>
</tr>
<tr>
<td>a</td>
<td>Toggle control registers display</td>
</tr>
<tr>
<td>c [&lt;num&gt; &lt;value&gt;]</td>
<td>Set a watchpoint trigger event counter. If no &lt;num&gt; nor &lt;value&gt; is specified, all the watchpoint trigger event counters will be printed. The SH8001 has one event counter numbered zero.</td>
</tr>
<tr>
<td>d</td>
<td>Toggle floating-point register display (decimal format)</td>
</tr>
<tr>
<td>f</td>
<td>Toggle floating-point register display (hex format)</td>
</tr>
<tr>
<td>k [&lt;watch_num&gt;]</td>
<td>Kill watchpoint(s)</td>
</tr>
<tr>
<td></td>
<td>&lt;watch_num&gt;: watchpoint number</td>
</tr>
<tr>
<td></td>
<td>If &lt;watch_num&gt; not present, kill all watchpoints</td>
</tr>
<tr>
<td>m</td>
<td>Toggle MMU registers display</td>
</tr>
<tr>
<td>tw</td>
<td>Trace over an operand access or instruction value watchpoint trigger.</td>
</tr>
<tr>
<td>w[&lt;ia</td>
<td>oa</td>
</tr>
</tbody>
</table>
Using RomBug

SH-5-specific command information is provided in the following table.

Table 9-3. 79R4700 Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Enable/disable monitoring of processor-specific default exception vectors. Below are the default vector numbers for exceptions: NMI= 14 Address error (instr)= 87 Address error (read)= 7 Address error (write)= 8</td>
</tr>
</tbody>
</table>

Supported Registers

The SH-5 architecture provides 64 64-bit general purpose registers, one 64-bit program counter register, 64 32-bit floating-point registers, 64 64-bit system control registers, and eight 64-bit target address registers.
General Purpose Registers

Table 9-4. General Purpose Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>RomBug Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0, r1</td>
<td></td>
<td>Caller-save registers</td>
</tr>
<tr>
<td>r2-r9</td>
<td></td>
<td>Incoming parameter registers, r2 is return value</td>
</tr>
<tr>
<td>r10</td>
<td>lr</td>
<td>Link register</td>
</tr>
<tr>
<td>r11</td>
<td>at</td>
<td>Assembler/linker temporary register</td>
</tr>
<tr>
<td>r12</td>
<td>fp</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>r13</td>
<td>cp</td>
<td>Constant data area pointer</td>
</tr>
<tr>
<td>r14</td>
<td>gp</td>
<td>Global data area pointer</td>
</tr>
<tr>
<td>r15</td>
<td>sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>r16-r19</td>
<td></td>
<td>Caller-save registers</td>
</tr>
<tr>
<td>r20-r23</td>
<td></td>
<td>Callee-save registers</td>
</tr>
<tr>
<td>r24-r31</td>
<td></td>
<td>Caller-save registers</td>
</tr>
<tr>
<td>r32-r62</td>
<td></td>
<td>Callee-save registers</td>
</tr>
<tr>
<td>r63</td>
<td>zero</td>
<td>Hard-coded zero value register</td>
</tr>
</tbody>
</table>

Program Counter Register

Table 9-5. SH-5 Program Counter Register

<table>
<thead>
<tr>
<th>Register Name</th>
<th>RomBug Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>pc</td>
<td>Program counter register</td>
</tr>
</tbody>
</table>
System Control Registers

There are 64 64-bit control registers.

<table>
<thead>
<tr>
<th>Register Number</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sr</td>
<td>Status register</td>
</tr>
<tr>
<td>1</td>
<td>ssr</td>
<td>Saved status register</td>
</tr>
<tr>
<td>2</td>
<td>pssr</td>
<td>Panic-saved status register</td>
</tr>
<tr>
<td>4</td>
<td>intenv</td>
<td>Interrupt event ID</td>
</tr>
<tr>
<td>5</td>
<td>expevt</td>
<td>Exception event ID</td>
</tr>
<tr>
<td>6</td>
<td>pexpevt</td>
<td>Panic-saved exception event ID</td>
</tr>
<tr>
<td>7</td>
<td>tra</td>
<td>TRAP exception number</td>
</tr>
<tr>
<td>8</td>
<td>spc</td>
<td>Saved program counter</td>
</tr>
<tr>
<td>9</td>
<td>pspc</td>
<td>Panic-saved saved program counter</td>
</tr>
<tr>
<td>10</td>
<td>resvec</td>
<td>Reset vector</td>
</tr>
<tr>
<td>11</td>
<td>vbr</td>
<td>Vector base register</td>
</tr>
<tr>
<td>13</td>
<td>tea</td>
<td>Faulting effective address</td>
</tr>
<tr>
<td>16</td>
<td>dcr</td>
<td>Debug control register</td>
</tr>
<tr>
<td>17</td>
<td>kcr0</td>
<td>Kernel control register 0</td>
</tr>
<tr>
<td>18</td>
<td>kcr1</td>
<td>Kernel control register 1</td>
</tr>
<tr>
<td>62</td>
<td>ctc</td>
<td>Clock tick counter</td>
</tr>
<tr>
<td>63</td>
<td>usr</td>
<td>User status register</td>
</tr>
</tbody>
</table>
Floating-point General Purpose Registers

Even numbered 32-bit adjacent floating-point general purpose registers (FR) can be paired and used as 64-bit floating-point general purpose registers (DR). RomBug will display both the 32-bit and 64-bit FPU registers.

<table>
<thead>
<tr>
<th>Register Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fr0 - fr11</td>
<td>Incoming floating-point parameters, fr0 and fr1 are floating-point return registers</td>
</tr>
<tr>
<td>fr12 - fr15</td>
<td>Callee-save registers</td>
</tr>
<tr>
<td>fr16 - fr35</td>
<td>Caller-save registers</td>
</tr>
<tr>
<td>fr36 - fr63</td>
<td>Callee-save registers</td>
</tr>
</tbody>
</table>

Target Address Registers

Eight 64-bit target address (TR) registers are provided:

<table>
<thead>
<tr>
<th>Register Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tr0 - tr4</td>
<td>Caller-save target address registers</td>
</tr>
<tr>
<td>tr5 - tr8</td>
<td>Callee-save target address registers</td>
</tr>
</tbody>
</table>

Display Information

The following register displays will be defined in this section:

- Normal
- Status
- Floating-point Status and Control

Normal Register Display

The normal register display for SH-5 is:

0000000000000000

Table 9-7. SH-5 Floating-point registers

Table 9-8. SH-5 Target address registers
Status Register Display

The status register (SR) is System Control Register number 0.

\[ \text{sr: 400080F0 (} -S------++asid++D-----mqimsk--s-) \]

Table 9-9. Status Register Bit Field Assignments

<table>
<thead>
<tr>
<th>Bit Field Number</th>
<th>Bit Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>Saturation control for SHcompact integer instructions (S = set, s = clear)</td>
</tr>
<tr>
<td>4 - 7</td>
<td>IMASK</td>
<td>Interrupt mask level (always displays imsk)</td>
</tr>
<tr>
<td>8</td>
<td>Q</td>
<td>Divide step state for SHcompact integer instructions (Q = set, q = clear)</td>
</tr>
<tr>
<td>9</td>
<td>M</td>
<td>Divide step state for SHcompact integer instructions (M = set, m = clear)</td>
</tr>
<tr>
<td>11</td>
<td>CD</td>
<td>Clock tick counter disable (C = set, - = clear)</td>
</tr>
<tr>
<td>12</td>
<td>PR</td>
<td>Precision for SHcompact floating-point instructions (P = set, - = clear)</td>
</tr>
<tr>
<td>13</td>
<td>SZ</td>
<td>Size for SHcompact floating-point instructions (S = set, - = clear)</td>
</tr>
<tr>
<td>14</td>
<td>FR</td>
<td>Floating-point register bank for SHcompact floating-point instructions (R = set, - = clear)</td>
</tr>
<tr>
<td>15</td>
<td>FD</td>
<td>Floating-point disable (D = set, - = clear)</td>
</tr>
<tr>
<td>16 - 23</td>
<td>ASID</td>
<td>Address space ID (Always displays ++asid++)</td>
</tr>
<tr>
<td>26</td>
<td>WATCH</td>
<td>Watchpoint enable flag (W = set, - = clear)</td>
</tr>
<tr>
<td>27</td>
<td>STEP</td>
<td>Single-step enable flag (S = set, - = clear)</td>
</tr>
<tr>
<td>28</td>
<td>BL</td>
<td>Event block flag (B = set, - = clear)</td>
</tr>
<tr>
<td>30</td>
<td>MD</td>
<td>Mode bit (S = set, system-state, U = clear, user-state)</td>
</tr>
<tr>
<td>31</td>
<td>MMU</td>
<td>MMU enable flag (M = set, - = clear)</td>
</tr>
</tbody>
</table>
Floating-point Status and Control Register (FCR31)

**Flag Bits:**
- Exception case detected, however not enabled.
- \( I = \) Inexact operation
- \( U = \) Underflow
- \( O = \) Overflow
- \( Z = \) Division by zero
- \( V = \) Invalid operation

**DN - Denormalized Bit:**
- 0 = Denorms cause FPU error
- 1 = Denorms treated as zero

**fpcsr:** 00000000 (--------------DEVZOUIVZOUIVZOU100)

**Cause bits:**
Written by each floating-point operation. If set, can cause an exception.
- \( E = \) FPU error

**Enable bits:**
- When set, corresponding bit causes appropriate exception.

**RM bit:**
- 0 = round to nearest number
- 1 = round toward

Rombug Examples

Setting Breakpoints

Setting breakpoints is done with the `b` command. An illustration of the command's usage follows. It sets a breakpoint at two labels: `dbg6` and `dbgR`.
<Called>

Trace Command

The following example illustrates the trace and memory display commands.

```
Chapter 9: SH-5 Processors

r52: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r56: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r60: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
tr0: 000012FD 000081FD 000081DD 00000000 00000000 00000000 00000000 00000000
sr: 400080F0 (-S--------+asid++D--------+mqimsk--s-)

pc: 000012A9
0x000012A8 >88000C10   ld.l r0,12,r1
trace: d5 .r0
0x400494D4 - A8900001 00000000 400498E0 0002B535 (......@..`..55
0x400494E4 - 0002C299 00000000 00000000 00000000 ..B............
0x400494F4 - 00000000 00000000 00000000 0002F461 ...............ta
0x40049504 - 400494D0 00000000 00000000 00000000 @..P............
dis: .rr1 .r0
RomBug: t
r00: 00000000400494d4 000000000002b535 0000000040041498 0000000040041000
r04: 0000000040041c50 0000000040041498 00000000400401000 0000000040041810
r08: 0000000000000000 0000000000000000 0000000000000000 00000000001295
r12: 0000000000000000 0000000000000000 0000000000000000 000000000007ff0
r16: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r20: 0000000040041810 0000000000000000 0000000000000000 0000000000000000
r24: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r28: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r32: 0000000040041000 0000000000000000 0000000000000000 000000000002400
r36: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r40: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r44: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r48: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r52: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r56: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
r60: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
tr0: 000012FD 000081FD 000081DD 00000000 00000000 00000000 00000000 00000000
sr: 400080F0 (-S--------+asid++D--------+mqimsk--s-)

pc: 000012AD
0x000012AC >CC0000B0  movi 0,r11
trace: d5 .r11
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
dis: