RadiSys.

Using RomBug

Version 5

www.radisys.com
World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision G
February 2001

Table of Contents

Chapter 1: 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.

For More Information

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
- Intel Pentium and 80x86 family
- Hitachi SuperH family
- MIPS
- Motorola 68k 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)?
```

Note

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.

Note

Relocation registers are considered symbols by RomBug and are present in displays.

Table 1-1 Relocation Register Commands

Command	Description
`@`	Print default relocation
`@<num>`	Set default relocation register to `<num>`. `<num> = 0 to 7.`

Table 1-1 Relocation Register Commands

Command		Description

`.r`
`.rr`
	†	Display relocation registers.

`.r<num> <val>`
`.rr<num> <val>`
	†	Set specified relocation register to `<val>`
		NOTE: Relocation register 0 is hard-wired to zero.

† `.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
0x00000108   >00000000         nop
0x0000010C   >00000000         nop
dis: @4                 /* make .r4 default relocatable register
*/
RomBug: .r4 0          /* set .r4 to zero */
RomBug: di 100 4     /* since .r4 is zero, no biasing is used
*/
0x00000100   >00000000         nop
0x00000104   >00000000         nop
0x00000108   >00000000         nop
0x0000010C   >00000000         nop
RomBug: a testprog
RomBug: .r4 download /* assign code symbol value to default
                        register */
RomBug: di download 4 /* name symbols are not biased by
                         relocatable */
download            >0DB0A0E1         mov r11,sp
download+0x4        >3F502DE9         stmdb sp!,{r0-r5,r12,lr}
download+0x8        >D0D04DE2         sub sp,sp,#0xD0
download+0xC        >00B08DE5         str r11,[sp]
RomBug: .r1 10      /* assign relocatable register .r1 the
                       value 10 */
RomBug: di .r1       /* .r4 is added to the value of .r1
                       before evaluation */
debug_entry+0x10    >000000EB         bl debug_entry+0x18-->
debug_entry+0x14    >84CDFFFF         swi 0xFFCD84
debug_entry+0x18    >00C09EE5         ldr r12,[lr]
debug_entry+0x1C    >0EC08CE0         add r12,r12,lr
```

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

Command	Description
`<symbol>`	Code or data symbol
`<num>`	`<num>` is interpreted as a number in the default radix
`#<num>`	`<num>` is a valid decimal (base 10) number
`0x<num>`	`<num>` is a valid hexadecimal (base 16) number
`<char>`	The ASCII value of `<char>` is sign extended to a 32-bit word.
`<reg>`	Valid register. For a complete list of machine registers used by supported processors, refer to the appropriate processor-specific chapter of this manual.

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

Operator	Function
`-e1`	Negate e1
`~e1`	Complement e1

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

Operator	Function
`e1 + e2`	Add e2 to e1
`e1 - e2`	Subtract e2 from e1
`e1 * e2`	Multiply e1 by e2
`e1 / e2`	Divide e1 by e2
`e1 > e2`	Bitwise right shift e1 by e2 bits
`e1 < e2`	Bitwise left shift e1 by e2 bits
`e1 & e2`	Bitwise AND of e1 and e2
`e1 | e2`	Bitwise OR of e1 and e2
`e1 ^ e2`	Bitwise exclusive OR of e1 and e2

Table 1-5 Indirect Operators

Operator	Function
`[e1]`	Uses the expression `e1` as the address of a long value for this term
`[e1]l`	Default. Uses the expression `e1` as the address of a long value for this term.
`[e1]w`	Uses the expression `e1` as the address of a word value which is sign-extended to 32-bits and used as the value for this term
`[e1]b`	Uses the expression `e1` as the address of a byte value which is sign-extended to 32-bits and used as the value for this term

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 1-6 Commands

Command	Description
`?`	Print Help
`@`	Print default relocation
`@<num>`	Set default relocation register to `<num>`. `<num> = 0 to 7.`
`a`	Attach to all modules found (after the system is up)
`a [<mod>]`	Attach to symbol module(s) for `<mod>`(s)
`am<beg><end>`	Attach to all modules found in address range `<beg>` to `<end>`
`b`	List breakpoints
`b <addr>`	Set breakpoint at `<addr>`

Table 1-6 Commands (continued)

Command	Description
`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): `l` 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

Table 1-6 Commands (continued)

Command	Description
`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): `l` 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)

Command	Description
`dd[<num>] [<expr>] [<len>]`	Double precision floating point display mode. Display memory at `<expr>` for `<len>` bytes as double 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:** Command not available on all processors.
`df[<num>] [<expr>] [<len>]`	Single precision floating point display mode. Display memory at `<expr>` for `<len>` bytes as single 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:** Command not available on all processors.

Table 1-6 Commands (continued)

Command	Description
`di[<num>] [<expr>] [<len>]`	Instruction disassembly mode. Disassemble instructions at `<expr>` for `<len>` instructions. 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.
`dn[e] <cmd>`	Download S-Record code. e echos S-Records. If *e* is not specified, load addresses are displayed every 512 bytes. `<cmd>` is issued to the shell to trigger the download. The I/O delay must be set in relocation register 1 before the download.

Table 1-6 Commands (continued)

Command	Description
`dx[<num>] [<expr>] [<len>]`	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.
`e`	Enable/disable monitoring of processor-specific default exception vectors (system state only)
`g`	Go at Program Counter (PC)
`g <addr>`	Go at `<addr>`
`gb`	Go with boot staging calls to the debugger
`gs`	Execute next instruction and stop
`gs <addr>`	Execute until `PC == <addr>`

Table 1-6 Commands (continued)

Command	Description
`k <addr>`	Kill breakpoint at `<addr>`
`k*`	Kill all breakpoints
`l <module>`	Link to memory module. Place address in `relocation register 7.`
`mc<dest><src><size>`	Copies memory from the address specified by `<src>` to the address specified by `<dest>`. The number of bytes to copy is specified by `<size>`.
`mf[<s>][n] <beg> <end> <value>`	Fill memory range with the pattern in `<value>`. `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 `<beg>` and `<end>` addresses.

Table 1-6 Commands (continued)

Command	Description
`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.

Table 1-6 Commands (continued)

Command	Description
`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.
`r [<module>]`	Remove symbols for `<module>`
`r*`	Remove all symbols
`rst`	Reset the system

Table 1-6 Commands (continued)

Command	Description
`s [mod:]<symb>`	Displays a single symbol from the current symbol module or symbol module `mod:`. The `*` and ? wildcard symbols may be used in the symbol name.
`sc <module>`	Shows code symbols for symbol module
`sd <module>`	Shows data symbols for symbol module (addresses based on current Global Data Register)
`sm`	Show symbol module directory
`ss <addr>`	Set default symbol module to module containing `<addr>`
`ss <name>:`	Set default symbol module to module `<name>`
`t [<num>]`	Trace one or <num> 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.
`tm<char>`	Talk mode. Escape from talk mode with `<char>.`
`v <expr>`	Print the value of the expression `<expr>` in hex and decimal

Table 1-6 Commands (continued)

Command	Description
`w[<num>]`	Print subroutine stack; `<num>` specifies the number of calls displayed
`x`	External OEM command
`.`	Print registers
`.<reg> <num>`	Set register `<reg>` to `<num>` **NOTE:** Relocation register 0 is hard-wired to zero. Reference the appropriate processor-specific chapter for processor-specific register information.
`.r` † `.rr`	Print relocation registers.
`.r<num> <val>` † `.rr<num> <val>`	Set specified relocation register to `<val>` **NOTE:** Relocation register 0 is hard-wired to zero.
`,`	Print floating point registers

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

Chapter 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 2-1 Breakpoint Commands

Command	Description
`b`	Display breakpoint list
`b<addr>`	Set breakpoint at the address specified by `<expr>`
`k<addr>`	Kill breakpoint at the address specified by `<addr>`
`k *`	Kill all breakpoints

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 2-2 Talk-Through Command

Command	Description
`tm <EscChar>`	Enter talk-through mode. This mode is exited when the specified escape character is typed.

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.

Note

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` special search table. There are two versions of the download command as identified in Table 2-3 Download Command.

Table 2-3 Download Command

Command	Description
`dl <HostCmd>`	Downloads data in Motorola S-record format. `<HostCmd>` 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.
`dle <HostCmd>`	Same as `dl <HostCmd>` except received S-records are also displayed on the console instead of load addresses.

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.

Note

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 2-4 Program Execution Commands

Command	Description
`g`	Go. Continues execution at the current PC until a RomBug monitored event is encountered.
`g <addr>`	Go from address. Continues execution at `<addr>` until a RomBug monitored event is encountered.
`gb`	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.
`gs`	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.
`gs <addr>`	Go and stop at address. Continues execution starting at the address in the PC register up to the specified `<addr>`. This is the same as the g command but it sets a breakpoint at `<addr>`. The breakpoint is automatically removed when the debugger regains control.

Table 2-4 Program Execution Commands (continued)

Command	Description
`t`	Trace. Traces one instruction and re-displays the machine registers.
`t <count>`	Trace instructions. Traces `<count>` instructions and re-displays the machine registers. Each instruction is displayed as it is executed. Breakpoints are ignored while tracing.

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>
dn:00005300 00000100 00000000 00020000 00000100 00000005 FFFFFFFF 00004300
an:FFF10378 FFF004B4 00005300 FFF80001 00005300 000042BC 00000400 00004292
pc:FFF00AAC sr:2704 (--SI-7--Z--)t:OFF msp:EC57A71C usp:00000400   ^isp^
0xFFF00AAC   >67F8            beq.b 0xFFF00AA6->
RomBug: g fff00aac                        go starting at fff00aac
                                          abort switch pressed
<Aborted>
dn:00005300 00000100 00000000 00020000 00000100 00000005 FFFFFFFF 00004300
an FFF10378 FFF004B4 00005300 FFF80001 00005300 000042BC 00000400 00004292
pc FFF00AAC sr:2704 (--SI-7--Z--)t:OFF msp:EC57A71C usp:00000400  ^isp^
0xFFF00AAC   >67F8            beq.b 0xFFF00AA6->
RomBug: gb                                go and boot the system
h320                                      hard disk VME320 boot selected from above
                                          menu
```

A valid OS-9 for 68K bootfile was found.

```
<Called>
dn:000FFC00 00000014 00000001 00000000 00000000 00000000 00000000 00000000
an:0000567E FFF004B4 00000000 00000000 00005300 00000400 00004300 000042F0
pc:FFF00882 sr:2708 (--SI-7-N---)t:OFF msp:EC57A71C usp:00000400   ^isp^
0xFFF00882   >142D0DE6         move.b 3558(a5),d2
RomBug: gb
```

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

CRC has not been verified.

```
<Called>
dn:0000049F 0000FFFF 000076C6 00000000 00000000 00000000 0000FFFF 00000000
an:FFF004C4 0000648C 000076F4 00004300 0000F600 00000DEC 00004300 0000FE00
pc:00005A02 sr:2708 (--SI-7-N---)t:OFF msp:EC57A71C usp:00000400   ^isp^
0x00005A02   >41FAFCB5         lea.l 0x56B9(pc),a0
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)

```
-tnxnp
tmode -w=1 nopause
link shell cio
load math881
SYS/startnet
*
* Load Network Modules and start Network
*
```

Abort switch pressed.

```
                                                 <Aborted>
dn:00000000 00002004 000EE118 000EDFA2 00000100 000EE104 000067C0 00840080
an:0000467C 000F6300 00004684 00004300 000ED960 000FE14C 00004300 00004300
pc:00007146 sr:2709 (--SI-7-N--C)t:OFF msp:EC57A71C usp:00012C64  ^isp^
0x00007146   >007C0700         ori.w #1792,sr
RomBug: t3
```

trace next three instructions

```
dn:00000000 00002004 000EE118 000EDFA2 00000100 000EE104 000067C0 00840080
an:0000467C 000F6300 00004684 00004300 000ED960 000FE14C 00004300 00004300
pc:0000714A sr:2709 (--SI-7-N--C)t:OFF msp:EC57A71C usp:00012C64   ^isp^
0x0000714A   >202C02E0         move.l 736(a4),d0
dn:7FFEFFBC 00002004 000EE118 000EDFA2 00000100 000EE104000067C0 00840080
an:0000467C 000F6300 00004684 00004300 000ED960 000FE14C 00004300 00004300
pc:0000714E sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00012C64   ^isp^
0x0000714E   >67D8             beq.b 0x7128
dn:7FFEFFBC 00002004 000EE118 000EDFA2 00000100 000EE104 000067C0 00840080
an:0000467C 000F6300 00004684 00004300 000ED960 000FE14C 00004300 00004300
pc:00007150 sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00012C64   ^isp^
0x00007150   >302E08AA         move.w 2218(a6),d0
trace:t
```

trace next instruction

```
dn:00000000 00002004 000DACB8 000DABC2 00000100 00000000 00000000 00840080
an:0000467C 000EC800 00004684 00004300 000DA500 000FE14C 00004300 00004300
pc:00007144 sr:2709 (--SI-7-N--C)t:OFF msp:EC57A71C usp:00013E7C   ^isp^
0x00007144   >67E2             beq.b 0x7128
```

```
trace: gs                                        go to next instruction
<at breakpoint>
dn:00000000 00002004 000DACB8 000DABC2 00000100 00000000 00000000 00840080
an:0000467C 000EC800 00004684 00004300 000DA500 000FE14C 00004300 00004300
pc:00007146 sr:2709 (--SI-7-N--C)t:OFF msp:EC57A71C usp:00013E7C   ^isp^
0x00007146   >007C0700         ori.w #1792,sr
trace: t
dn:00000000 00002004 000DACB8 000DABC2 00000100 00000000 00000000 00840080
an:0000467C 000EC800 00004684 00004300 000DA500 000FE14C 00004300 00004300
pc:0000714A sr:2709 (--SI-7-N--C)t:OFF msp:EC57A71C usp:00013E7C   ^isp^
0x0000714A   >202C02E0         move.l 736(a4),d0
trace: gs 7146                                   go and stop at hex 7146
<at breakpoint>
dn:00000000 00002004 000DACB8 000DABBA 00000100 00000000 00000000 00840080
an:0000467C 000EC800 00004684 00004300 000ED960 000FE14C 00004300 00004300
pc:00007146 sr:2719 (--SI-7XN--C)t:OFF msp:EC57A71C usp:00013E7C   ^isp^
0x00007146   >007C0700         ori.w #1792,sr
```

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

Command	Description
`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)

Note

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
$00020000   4F: 'a
$00020001   6F: 20
$00020002   6F: -
$00020001   20: #20
$00020002   6F: -
$00020001   14: +
$00020002   6F: .
RomBug: d1 .d3
$00020000    - 61146F68 42616279 4F6F6F68 42616279  a.ohBabyOoohBaby
dis: c1 .d3
```

```
$00000000    - 00008D00 10E0054C 10E1293A 10E12944  .....`.L.a):.a)D
$00000010    - 10E1294E 0000041E 00000428 00000432  .a)N.......(...2
$00000020    - 0000043C 10E12958 00000450 0000045A  ...<.a)X...P...Z
dis: d1 .d3
$00020000    - 61146F68 42616279 4F6F6F68 42616279  a.ohBabyOoohBaby
dis: cl .d3
$00020000   61146F68: #10000
$00020004   42616279: #44
$00020008   4F6F6F68: .
RomBug: d1 .d3
$00020000    - 00002710 0000002C 4F6F6F68 42616279  ..'....,OoohBaby
dis: cl .d3 #10000 #44 .
RomBug: d1 .d3
$00020000    - 00002710 0000002C 4F6F6F68 42616279  ..'....,OoohBaby
dis:
```

Memory Disassembly

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

Table 2-6 di Command

Command	Description
`di <addr> [<lines>]`	Disassembles and displays the specified number of machine instructions `<lines>` of memory starting at the address specified by `<addr>`. If `<lines>` is not specified, 16 lines of machine instructions are disassembled and displayed. `<addr>` must evaluate to an address that conforms to the processor-specific instruction alignment.

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).

Note

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.

Examples

```
RomBug: di main
main                >4E550000         link.w a5,#$0
main+$4             >48E7CCA0         movem.l d0-d1/d4-d5/a0/a2,-(a7)
main+$8             >41EE0E72         lea.l rest_env(a6),a0
main+$C             >2008             move.l a0,d0
main+$E             >61FF00009B36     bsr.l setjmp
main+$14            >2A00             move.l d0,d5
main+$16            >6710             beq.b main+$28
main+$18            >7201             moveq.l #$1,d1
main+$1A            >2005             move.l d5,d0
main+$1C            >61FFFFFFD2BE     bsr.l put_exception
main+$22            >95CA             suba.l a2,a2
main+$24            >600000DA         bra.w main+$100
main+$28            >61FFFFFFF1C2     bsr.l get_vectors
main+$2E            >42AE0E06         clr.l call_debug(a6)
main+$32            >95CA             suba.l a2,a2
main+$34            >61FF000015B4     bsr.l ConsSet
dis: di main 3
main                >4E550000         link.w a5,#$0
main+$4             >48E7CCA0         movem.l d0-d1/d4-d5/a0/a2,-(a7)
main+$8             >41EE0E72         lea.l rest_env(a6),a0
```

Memory Display

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

Note

Endianess is preserved in a display memory command.

Table 2-7 d Command

Command	Description
`d[{<size>[s]|<M>}][<num>] <addr> [<len>]`	`<addr>` is the starting address for the memory display. This value must be even for floating point memory display.

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 endianess for word and long access. Valid combinations:

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.

Note

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

Hex/ASCII Dump Memory Display

Note

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 (.).

```
dis: d main
main            - 4E550000 48E7CCA0 41EE0E72 200861FF  NU..HgL An.r .a.
main+$10        - 00009B36 2A006710 72012005 61FFFFFF  ...6*.g.r. .a...
main+$20        - D2BE95CA 600000DA 61FFFFFF F1C242AE  R>.J‘..Za...qBB.
main+$30        - 0E0695CA 61FF0000 15B44AAE 0E146732  ...Ja....4J...g2
main+$40        - 61FF0000 25427200 202E0E14 61FF0000  a...%Br. ...a...
main+$50        - 234C61FF 00002530 42AE0E14 42AE0E26  #La...%0B...B..&
main+$60        - 700061FF 000050CA 61FFFFFF BD1C6000  p.a...PJa...=.‘.
main+$70        - 00904AAE 0E266730 4A6E0E12 661C61FF  ..J..&g0Jn..f.a.
main+$80        - 00002504 4AAE0E1C 67107200 202E0E1C  ..%.J...g.r. ...
main+$90        - 61FF0000 230842AE 0E1C202E 0E4261FF  a...#.B... ..Ba.
main+$A0        - 00004FCC 24406058 4A6E0E12 660661FF  ..OL$@‘XJn..f.a.
main+$B0        - 000024D4 4AAE0E18 671A7201 202E0E18  ..$TJ...g.r. ...
main+$C0        - 61FF0000 243242AE 0E187001 2D400E26  a...$2B...p.-@.&
main+$D0        - 700060CA 700061FF FFFFDCDE 6022222E  p.‘Jp.a...\^‘"".
main+$E0        - 240241FA 03502008 61FFFFFF A82C223C  $.Az.P .a...(,"<
main+$F0        - 00000100 45EE11B8 200A61FF 000012DE  ....En.8 .a....^
dis: d main 44
main            - 4E550000 48E7CCA0 41EE0E72 200861FF  NU..HgL An.r .a.
main+$10        - 00009B36 2A006710 72012005 61FFFFFF  ...6*.g.r. .a...
main+$20        - D2BE95CA 600000DA 61FFFFFF F1C242AE  R>.J‘..Za...qBB.
main+$30        - 0E0695CA 61FF0000 15B44AAE 0E146732  ...Ja....4J...g2
main+$40        - 61FF0000 25427200 202E0E14 61FF0000  a...%Br. ...a...
dis: d5 main
main            - 4E550000 48E7CCA0 41EE0E72 200861FF  NU..HgL An.r .a.
main+$10        - 00009B36 2A006710 72012005 61FFFFFF  ...6*.g.r. .a...
main+$20        - D2BE95CA 600000DA 61FFFFFF F1C242AE  R>.J‘..Za...qBB.
main+$30        - 0E0695CA 61FF0000 15B44AAE 0E146732  ...Ja....4J...g2
main+$40        - 61FF0000 25427200 202E0E14 61FF0000  a...%Br. ...a...
dis: d1 main
main            - 4E550000 48E7CCA0 41EE0E72 200861FF  NU..HgL An.r .a.
dis: d main 1
main            - 4E550000 48E7CCA0 41EE0E72 200861FF  NU..HgL An.r .a.
```

Example Floating Point Memory Displays

```
dis: df 20200
$00020200    - 40490FDB 3.141592741012573
dis: dd 20000
$00020000    - 400921FB54442D18 3.141592653589793
dis: dx 20100
$00020100    - 40000000C90FDAA22168C235 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.

Change Machine Registers

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

Table 2-8 Change Machine Register Commands

Command	Description

`.<reg> <val>`	Changes the specified register to `<val>`. If a floating point register is specified, the change value may be either a double precision decimal constant or a left-justified hexadecimal value:
	`.<fpreg> <float-decimal constant>` or
	`.<fpreg> <left-justified hex constant>` or
	`.<fpreg> .<fpreg>`
	The syntax for `<float-decimal constant>` is:
	`[±]digits[.digits][Ee[±]integer]`
	The syntax for `<left-justified hex constant>` is:
	`0x<hexdigits>`

Note

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 2-9 mf Command

Command	Description
`mf [<s>] [n] <start> <end> <value> [<value>]`	The `mf` command is used to fill memory with a given pattern. The fill pattern size 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 fill is to be performed without regard to word/longword boundaries (word and longword fills are done on a byte for byte basis).

Note

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 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    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070010    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070020    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
dis: mfb 70000 70003 a1 fill with byte
RomBug: d2 70000
0x00070000    - A1A1A1A1 FEEDC0DE FEEDC0DE FEEDC0DE  !!!!~m@^~m@^~m@^
0x00070010    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
dis: mfw 70000 7000f 5252 fill with word
RomBug: d2 70000
0x00070000    - 52525252 52525252 52525252 52525252  RRRRRRRRRRRRRRRR
0x00070010    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
dis: mfl 70000 7000f 81186226 fill with longword
RomBug: d3 70000
0x00070000    - 81186226 81186226 81186226 81186226  ..b&..b&..b&..b&
0x00070010    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070020    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
dis: mfwn 70001 70007 0102 non-aligned fill word
RomBug: d3 70000
0x00070000    - 81010201 02010201 81186226 81186226  ..........b&..b&
0x00070010    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070020    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
dis: mfb 70000 7000f "shazam! fill with a string
RomBug: d4 70000
0x00070000    - 7368617A 616D2173 68617A61 6D217368  shazam!shazam!sh
0x00070010    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070020    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070030    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070040    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070050    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070060    - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
```

```
0x00070070     - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
dis: mf 70000 70010 1 2 3 4 3+2 fill with multiple byte values
RomBug: d4 70000
0x00070000     - 01020304 05010203 04050102 03040501  ................
0x00070010     - 02EDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  .m@^~m@^~m@^~m@^
0x00070020     - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
0x00070030     - FEEDC0DE FEEDC0DE FEEDC0DE FEEDC0DE  ~m@^~m@^~m@^~m@^
```

Memory

Table 2-10 ms Command

Command	**Description**
`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

Note

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.

Examples

```
RomBug: d1 btext
btext           - 4AFC0001 0000259E 000C0064 00000048  J|....%....d...H
dis: msw  btext btext+259e 1                          search for word-aligned 0001
btext+0x2       - 00010000 259E000C 00640000 00480555  ....%....d...H.U
sprintf+0x1C0   - 0001206F 000C58AF 000C2008 56802200  .. o..X/.. .V.".
putc+0x42       - 000141EF 00072208 306A000E 2008206A  ..Ao..".0j.. . j
fclose+0x20     - 0001670E 200A6126 28006006 4A6A000C  ..g. .a&(.‘.Jj..
fseek+0xB8      - 000167BC 0CAF0000 00010020 6608202A  ..g<./..... f. *
fseek+0xC0      - 00010020 6608202A 00089092 988024AA  ... f. *......$*
ftell+0x2E      - 00017200 306A000E 20086100 057C588F  ..r.0j.. .a..|X.
```

```
_setbase+0x2A    - 00016606 303C0080 60027040 48C0816A  ..f.0<..‘.p@H@.j
_setbase+0xC8    - 00010012 603E306A 00122008 22000CAE  ....‘>0j.. ."...
_iobinit+0x16    - 00018182 3D7C0002 819C3D7C 0002819E  ....=|....=|....
_T$LDiv+0x14     - 00016122 E20A6402 4480E20A 64024481  ..a"b.d.D.b.d.D.
getstat+0xC      - 0001672E 0C010006 673A0C01 00026710  ..g.....g:....g.
getstat+0x26     - 00016000 03044E40 008D6500 02FC206F  ..‘...N@..e..| o
lseek+0xC        - 00016716 0C010002 670672CB 6000013C  ..g.....g.rK‘..<
_utinit+0x2      - 00014E75 4E752F05 7A004A80 6A047A08  ..NuNu/.z.J.j.z.
Tens16+0xA6      - 00013C67 0EF54646 D4973C9C D2B297D8  ..<g.uFFT.<.R2.X
_T$DInt+0x3A     - 00015247 E288E291 51CEFFF2 64125281  ..RGb.b.QN.rd.R.
_T$DMul+0x18E    - 000108C0 001F6030 0C828000 00006604  ...@..‘0......f.
PackD+0x58       - 00010101 01010101 01011111 01111101  ................
PackD+0xE0       - 000104E8 00000000 00000001 04E40000  ...h.........d..
PackD+0xEA       - 000104E4 00000000 0085230F 0D74000F  ...d......#..t..

RomBug: msw  btext btext+259e 4e40          search for system calls
_stkchec+0x30    - 4E40008C 321F4E75 202E8000 90AE8008  N@..2.Nu .......
trapinit+0x1A    - 4E400021 64066100 12CE6564 2F490014  N@.!d.a..Ned/I..
trapinit+0xBA    - 4E400006 4E400006 12D866FC 4E7548E7  N@..N@...Xf|NuHg
trapinit+0xBE    - 4E400006 12D866FC 4E7548E7 C080203C  N@...Xf|NuHg@. <
getstat+0x2C     - 4E40008D 650002FC 206F0010 20826000  N@..e..| o.. .‘.
getstat+0x3E     - 4E40008D 650002EA 20016000 02E2206F  N@..e..j ..‘..b
ogetstat+0x50    - 4E40008D 600002D6 48E76080 C1414A81  N@..‘..VHg‘.AAJ.
setstat+0x1E     - 4E40008E 600002B0 48E76080 20403001  N@..‘..0Hg‘. @0.
access+0x8       - 4E400084 650002A2 4E40008F 60000294  N@..e.."N@..‘...
access+0x10      - 4E40008F 60000294 48E76080 20403001  N@..‘...Hg‘. @0.
open+0xA         - 4E400084 60000286 48E76080 4E40008F  N@..‘...Hg‘.N@..
close+0x4        - 4E40008F 6000027A 48E76080 20403001  N@..‘..zHg‘. @0.
mknod+0xA        - 4E400085 6500026A 60000260 48E76080  N@..e..j‘..‘Hg‘.
create+0x10      - 4E400083 6000023A 48E76080 2F092040  N@..‘..:Hg‘./. @
creat+0x16       - 4E400083 2049225F 64000218 0C0100DA  N@.. I"_d......Z
creat+0x36       - 4E400084 650001FE 74007202 4E40008E  N@..e..~t.r.N@..
creat+0x42       - 4E40008E 640001F0 34014E40 008F3202  N@..d..p4.N@..2.
creat+0x4C       - 4E40008F 32026000 01E648E7 60802040  N@..2.‘..fHg‘. @
unlinkx+0x8      - 4E400087 650001D6 600001CC 48E76080  N@..e..V‘..LHg‘.
unlink+0x8       - 4E400087 650001C2 600001B8 48E76080  N@..e..B‘..8Hg‘.
dup+0x4          - 4E400082 600001B0 48E76080 2041222F  N@..‘..0Hg‘. A"/
read+0xA         - 4E400089 65062001 6000019A 0C4100D3  N@..e. .‘....A.S
readln+0xA       - 4E40008B 60DC48E7 60802041 222F0010  N@..‘\Hg‘. A"/..
write+0xA        - 4E40008A 6500016E 20016000 016648E7  N@..e..n .‘..fHg
 writeln+0xA      - Strings must begin with a quotation mark (“).
                    E40008C 60E648E7 6080122F 00136726  N@..‘fHg‘../..g&
 lseek+0x1E       - 4E40008D 640E60F2 72054E40 008D6404  N@..d.‘rr.N@..d.
 lseek+0x28       - 4E40008D 640460E8 7400D497 22024E40  N@..d.‘ht.T.".N@
 lseek+0x36       - 4E400088 65DC2001 60000114 48E76080  N@..e\.‘...Hg‘.
 ebrk+0x42        - 4E400028 204A245F 650000C8 2D4884D8  N@.( J$_e..H-H.X
 sbrk+0x14        - 4E400007 64000008 225F6000 00642D40  N@..d..."_‘..d-@
 _srqmem+0x2      - 4E400028 650A2D40 84E0200A 245F4E75  N@.(e.-@.‘ .$_Nu
 _srtmem+0x8      - 4E400029 245F6000 000691C8 C1886406  N@.)$_‘....HA.d.
 _exit+0x2        - 4E400006 DEADDEAD 003C0001 4E754E75  N@..^-^-.<..NuNu
```

```
RomBug: msl  btext btext+259e :ffffff80 4e400000 only non-I/O calls
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: msln 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@..N@...Xf|NuHg
trapinit+0xBE    - 4E400006 12D866FC 4E7548E7 C080203C  N@...Xf|NuHg@. <
ebrk+0x42        - 4E400028 204A245F 650000C8 2D4884D8  N@.( J$_e..H-H.X
sbrk+0x14        - 4E400007 64000008 225F6000 00642D40  N@..d..."_'..d-@
_srqmem+0x2      - 4E400028 650A2D40 84E0200A 245F4E75  N@.(e.-@.' .$_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....."<...@/

dis: ms   btext  btext+259e "*             string search
trapinit+0x34    - 2A2A2A2A 20537461 636B204F 76657266  **** Stack Overf
trapinit+0x35    - 2A2A2A20 53746163 6B204F76 6572666C  *** Stack Overfl
trapinit+0x36    - 2A2A2053 7461636B 204F7665 72666C6F  ** Stack Overflo
trapinit+0x37    - 2A205374 61636B20 4F766572 666C6F77  * Stack Overflow
trapinit+0x48    - 2A2A2A2A 0D002A2A 2A2A2043 616E2774  ****..**** Can't
trapinit+0x49    - 2A2A2A0D 002A2A2A 2A204361 6E277420  ***..**** Can't
trapinit+0x4A    - 2A2A0D00 2A2A2A2A 2043616E 27742069  **..**** Can't i
                              .
                              .
                              .
_T$DMul+0x150    - 2A04CA81 84852A04 4685C285 C8808284  *.J...*.F.B.H...
_T$DMul+0x156    - 2A044685 C285C880 8284C085 4A826A3C  *.F.B.H...@.J.j<
_T$DDiv+0x64     - 2A3C0000 07FF2801 C885B981 80842803  *<....(.H.9...(.
NormD+0x66       - 2A044685 C285C882 82846022 E8B9E8B8  *.F.B.H...'"h9h8

RomBug: ms  btext  btext+259e 2a 00        same as previous but null terminated
sprintf+0x3B8    - 2A0047EE 803E2004 720FC081 2C002200  *.Gn.> .r.@.,.".
putc+0x7         - 2A000C48 C0028000 0080220C 80000080  *..H@.....".....
putc+0x1B        - 2A000C48 C07222C0 810C8000 00000266  *..H@r"@.......f
putc+0x35        - 2A000C48 C0080000 02672A48 78000141  *..H@....g*Hx..A
putc+0x6B        - 2A000C48 C0080000 08660620 0A610001  *..H@....f. .a..
putc+0x8D        - 2A000C48 C0080000 0766160C 97000000  *..H@....f......
                              .
                              .
                              .
fclose+0xD       - 2A000C48 C0080000 0F671430 2A000C48  *..H@....g.0*..H
fclose+0x19      - 2A000C48 C0080000 01670E20 0A612628  *..H@....g. .a&(
_setbase+0x4B    - 2A000C48 C0080000 07670E41 FA049E25  *..H@....g.Az..%
_setbase+0x75    - 2A000C48 C0720CC0 81660000 9E4A6A00  *..H@r.@.f...Jj.
_setbase+0x89    - 2A000C48 C0080000 07670835 7C020000  *..H@....g.5|...
_setbase+0xFB    - 2A001248 C0D1AE81 54700848 C0816A00  *..H@Q..Tp.H@.j.
_setbase+0x10D   - 2A000432 2A001248 C1D08125 40000824  *..2*..HAP.%@..$
_setbase+0x111   - 2A001248 C1D08125 40000824 80588F4C  *..HAP.%@..$.X.L
RomBug:
```

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

Command	Description
`mc<dest><src> <size>`	Copies memory from the address specified by `<src>` to the address specified by `<dest>`. The number of bytes to copy is specified by `<size>`.

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  ................
$002000A0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000B0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000C0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000D0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000E0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000F0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
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  ................
$00200070     - 01010101 01010101 01010101 01010101  ................
$00200080     - 01010101 01010101 01010101 01010101  ................
$00200090     - 01010101 01010101 01010101 01010100  ................
$002000A0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000B0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000C0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000D0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000E0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
$002000F0     - 4F6F6F68 42616279 4F6F6F68 42616279  OoohBabyOoohBaby
```

Linking to a Module

The `l` command is used to link to a memory module. The address of the module is placed in relocation register 7. Subsequent `l` commands unlink the previously linked module and link to the new module.

Note

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.

Table 2-12 l Command

Command	Description
`l <module>`	Links debugger to the module specified by `<module>`

Examples

```
RomBug: l term                                      link to the module named TERM
RomBug: @7                                          set default relocation register to 7
RomBug: .r                                           display relocation registers
Rn: 00000000 00000000  00000000 0001FE00   00000000 00000000  00000000 00004EB8
RomBug: d8 0                                             display memory
0x00000000+r7 - 4AFC0001 0000007A 00000000 00000070  J|.....z.......p
0x00000010+r7 - FFFF0F00 80000004 00000000 00000000  ................
0x00000020+r7 - 00000000 00000000 00000000 0000C5F3  ..............Es
0x00000030+r7 - 00FF8080 1B030123 00640068 00000000  .......#.d.h....
0x00000040+r7 - 00000000 0000001C 00000100 01010001  ................
0x00000050+r7 - 1808180D 1B040117 03050807 000F0070  ...............p
0x00000060+r7 - 11130904 53636600 73633638 36383100  ....Scf.sc68681.
0x00000070+r7 - 7465726D 00000022 24014AFC 00010000  term..."$.J|....
```

Symbolic Debugging

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

Table 2-13 Symbol Table Commands

Command	Description
`s`	Displays all symbols in all symbol modules
`s [mod:]<symb>`	Displays a single symbol from the current symbol module or symbol module `[mod:]`. The `*` and `?` wildcard symbols may be used in the symbol name.
`sm`	Displays symbol module table
`ss`	Sets current symbol module to the module containing the current PC
`ss <addr>`	Sets current symbol module to the module containing `<addr>`
`ss <name>:`	Sets current symbol module to the symbol module specified by `<name>`
`sd <name>`	Displays data symbols only for the specified symbol module
`sc <name>`	Displays code symbols only for the specified symbol module

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) and the absolute address of the symbol.

Note

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

```
RomBug: a ram                               attach to RAM disk driver
RomBug: s                                   display all symbols
ram:btext           C 0000B8C8   ram:Init            C 0000B938
ram:end             D 00010090   ram:Move            C 0000BA7C
ram:ReadSect        C 0000BA90   ram:WritSect        C 0000BA9C
ram:Seek            C 0000BAAC   ram:etext           C 0000BAB0
ram:PutStat         C 0000BAC8   ram:GetStat         C 0000BAC8
ram:Termnt          C 0000BAD2   ram:bname           C 0000BAEC
RomBug: sm                                  show symbol module table
Mod Addr   Code Lo  Code Hi  Data Lo  Data Hi Count  Name
001f0700  0000b8c8 0000baf2 fffffffe 0000008e    12  ram
RomBug: b ReadSect                          set breakpoint at ReadSect
RomBug: sm                                  show modules (current (*) is ram)
Mod Addr   Code Lo  Code Hi  Data Lo  Data Hi Count  Name
001f0700  0000b8c8 0000baf2 fffffffe 0000008e    12 *ram
RomBug: a kernel                            attach to kernel module
RomBug: s                                   show all symbols
```

```
btext                C 0000B8C8   Init                 C 0000B938
end                  D 00010090   Move                 C 0000BA7C
ReadSect             C 0000BA90   WritSect             C 0000BA9C
Seek                 C 0000BAAC   etext                C 0000BAB0
PutStat              C 0000BAC8   GetStat              C 0000BAC8
Termnt               C 0000BAD2   bname                C 0000BAEC
kernel:end           D 0000FFFE   kernel:btext         C 00003000
kernel:_syscmmt      D 0000FFFE   kernel:CopyRight     C 0000303C
kernel:MPUtype       C 0000307A   kernel:SysErMsg      C 0000316A
kernel:PCMsg         C 00003193   kernel:ResetMsg      C 0000319F
kernel:EOL           C 000031B4   kernel:Cold          C 000031B8
                      .
                      .
                      .
kernel:Wait          C 00006C0C   kernel:ChildSts      C 00006C62
kernel:etext         C 00006C6C   kernel:RemChild      C 00006C76
kernel:RetProc       C 00006C96   kernel:DelPrc        C 00006C96
kernel:bname         C 00006CA8
RomBug: sm                                  show symbol modules
Mod Addr   Code Lo  Code Hi  Data Lo  Data Hi Count  Name
00006cb4  00003000 00006cb2 fffffffe fffffffc   161  kernel
001f0700  0000b8c8 0000baf2 fffffffe 0000008e    12 *ram
RomBug: s ReadSect                          find a particular symbol
ReadSect             C 0000BA90
RomBug: s kernel:Cold                       find a symbol qualified with module name
kernel:Cold          C 000031B8
RomBug: s R*                                find symbol using wildcardReadSect  C
0000BA90   kernel:ResetMsg     C 0000319F
kernel:RtrnPD        C 000060BC   kernel:R64           C 000060BC
kernel:RetTime       C 00006754   kernel:RemChild      C 00006C76
kernel:RetProc       C 00006C96
RomBug: ss                                  set default symbols to 'RAM'
default symbols belong to 'ram'
```

OS-9 Example

```
RomBug: s
bname                C 00000000   bdata                D 00000000
vectbl               D 00000000   sysmem               D 00000400
totmem               D 00000404   cputype              D 00000408
datasize             D 0000040C   berrenv              D 00000410
aderrenv             D 00000450   romblks              D 00000490
dtype                D 00000494   inlptr               D 00000496
dnum                 D 0000049A   fp_cir               D 0000050A
mmu                  D 000007B2   regs                 D 00000818
fregs                D 00000884   stat_reg_save        D 000008F2
vf                   D 000008F4   break_flag_save      D 000009F4
vect_tab             D 000009F6   vectors              D 000009FA
irqdone              D 00002EC0   scsi33c93_usecnt     D 00002EC4
_jmptbl              D 00002EC8   con8530a             D 00002FA6
con8530b             D 00002FDA   end                  D 0000302A
dsize                C 00005100   btext                C 003C0000
```

```
initial_vectors     C 003C0000   cold                C 003C0010
sysreturn           C 003C0014   memclear            C 003C0022
memsearch           C 003C005A   buserr              C 003C006E
addrerr             C 003C00A6   getcpu              C 003C00B8
fatalerr            C 003C0130   ilglexcp            C 003C0134
RomBug: sm
Mod Addr   Code Lo  Code Hi  Data Lo  Data Hi Count  Name
003d399a  003c0000 003d298e 00010000 0001a200   544 *boot
RomBug: s main
main                C 003C612A
RomBug: s boot:main
main                C 003C612A
```

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 2-14 Attach Module Commands

Command	Description
`a [<module>]`	Attaches to the specified module
`am <beg> <end>`	Attaches all modules found in the address range specified by `<beg>` to `<end>`

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 symbolic and associated code modules found in the specified address range.

Note

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               attaches to all modules found in the module directory
RomBug: a rb1772        attaches to module rb1772
RomBug: a rbf rb1772    attaches to module rbf and rb1772
RomBug: am 70000 90000  attaches all symbol and code modules in range
```

Viewing Expressions

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

Table 2-15 v Command

Command	Description
`v <expr>`	Evaluates an expression and prints the value in decimal, hexadecimal, and as a symbolic address

Example 1

```
dn: 0000000C 000C0064 00000080 00000003 00000000 000000A2 00001050 00000000
an: 00000000 00015F80 00000000 000F32A0 00000000 00015EDC 0001CF30 00015EDC
pc: 000F32EE  cc: 00 (-----)
<68881 in Null state>
_cstart             >2D468010         move.l d6,_totmem(a6)
RomBug: v .d1
0x000C0064 (786532) 0xC0064
RomBug: v .pc
0x000F32EE (996078) _cstart
RomBug: v .pc+10
0x000F32FE (996094) _cstart+0x10
RomBug: v .pc+400
0x000F36EE (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 00014F4A 00015F56 00000000 000000A2 00001050 00000000
an: 000152DE 001F214A 00015F4E 00015F46 00015F42 00000000 0001CF30 00015ED4
pc: 000F34F0  cc: 00 (-----)
<68881 in Null state>
main                >48E7F080         movem.l d0-d3/a0,-(a7)
RomBug: v .a1
0x001F214A (2040138) cio:CIOTrap
RomBug: v .d1
```

```
0x00015F4E (89934) 0x15F4E
RomBug: v [.d1]
0x00015F46 (89926) 0x15F46
RomBug: d1 [.d1]
0x15F46              - 70726F67 78000000 00015F46 00000000  progx....._F....
```

Example 2

```
dn:00000000 00012345 00000000 0000004A 00000000 0000FA08 00000000 00000000
an:00008164 00000000 00000045 00000000 00000000 00000000 00002044 00000000
pc: 00000000  cc: 00 (-----)
<68881 in Null state>
dis: v .d1
0x00012345 (74565) 0x00012345
RomBug: v .d5>3
0x00001F41 (8001) 0x00001F41
RomBug: v 11+69
0x0000007A (122) 0x0000007A
RomBug: v #11+#69       explicit decimal numbers
0x00000050 (80) 0x00000050
RomBug: v fe61*2        some hex values confuse debug
Symbol 'fe61*2' not found
RomBug: v 0xfe61*2      use 0x for clarity
0x0001FCC2 (130242) 0x0001FCC2
RomBug: v .a0-24
0x00008140 (33088) 0x00008140
RomBug: v .a0&fff
0x00000164 (356) 0x00000164
RomBug: v .d5-5*.a2
0x004362CF (4416207) 0x004362CF
RomBug: v .d5-(5*.a2)
0x0000F8AF (63663) 0x0000F8AF
RomBug: v .a2^.d3
0x0000000F (15) 0x0000000F
RomBug: d1 .a6
0x00002044    - 001FEF70 001FF4F4 00156440 000023E4  ..op..tt..d@..#d
dis: v [.a6]            memory indirection
0x001FEF70 (2092912) 0x001FEF70
RomBug: v [.a6]+10
0x001FEF80 (2092928) 0x001FEF80
RomBug: v [.a6]w
0x0000001F (31) 0x0000001F
RomBug: v [.a6]w+1
0x00000020 (32) 0x00000020
RomBug: v [.a6]b
0x00000000 (0) 0x00000000   Command
```

Chapter 3: 68xxx Processors

This chapter includes information on 68xxx processor-specific information.

-o Option

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

Table 3-1 68K Options

Option	Description
a	Toggle control registers
`c<n>[:f]`	Set MPU type to <n> where <n> = processor number and FPCP to `6888<f>`
d	Toggle FPCP decimal register display
`e<addr>`	Display exception frame (default `<addr>` is `.a7`)
`f`	Toggle FPCP register display
`m`	Toggle MMU register display

Commands

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

Table 3-2 Commands

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

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 3-3 68xxx Registers

Register	Description
`.d0 - .d7`	Data registers
`.a0 - .a7`	Address registers
`.usp`	User stack pointer
`.ssp`	Supervisor stack pointer
`.sp`	Current stack pointer
`.cc`	Condition code register
`.sr`	Status register
`.pc`	Program counter

Table 3-4 68010/20/30/40/60 Only Registers

Register	Description
`.vbr`	Vector base register
`.sfc`	Source function code register
`.dfc`	Destination function code register
`.cacr`	Cache control register

Table 3-5 68020/30/40 Only Registers

Register	Description
`.msp`	Master stack pointer
`.isp`	Interrupt stack pointer
`.caar`	Cache address register

Table 3-6 68030/68551 Only Registers

Register	Description
`.crp`	CPU root pointer
`.srp`	Supervisor root pointer
`.tc`	Translation control register

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

Register	Description
`.fp0 - .fp7`	Floating point data registers
`.fpsr`	Floating point status register
`.fpcr`	Floating point control register
`.fpiar`	Floating point instruction address register

Table 3-8 68030 Only Registers

Register	Description
`.tt0`	Transparent translation register
`.tt1`	Transparent translation register
`mmusr`	MMU status register

Table 3-9 68040/60 Only Registers

Register	Description
`.dtt1`	Data transparent translation register
`.dtt0`	Data transparent translation register
`.itt1`	Instruction transparent translation register
`.itt0`	Instruction transparent translation register
`.urp`	User root pointer register
`.srp`	Supervisor root pointer register
`.tcr`	Translation control register

Table 3-10 68060 Only Registers

Register	Description
`.pcr`	Processor configuration register
`.buscr`	Bus control register

Table 3-11 68551 Only Registers

Register	Description
`.drp`	DMA root pointer register
`.pcsr`	PMMU cache status register
`.psr`	PMMU status register
`.scc`	Stack change control register
`.ac`	Access control register

Display Information

Register information shown below displays following an oa command.

```
vbr:00000000 sfc:7=CS dfc:7=CS      cacr:1=.E  caar:80000000
dn:0000000D 0001DE3A 00000001 00000003 00000000 000000D6 01010101 00000000
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>
```

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 `pcr` and `buscr` registers. The oa option toggles the display of this line. The first three registers are displayed for the 68010/20/30/40/60/CPU32. `vbr` is the vector base register. `sfc` and `dfc` are the source and destination function code registers, respectively. The three bit value of the `sfc/dfc` registers is interpreted as identified in the following table.

Table 3-12 sfc/dfc Register Three Bit Value

Value	Text	Meaning
0	??	Undefined
1	UD	User data space
2	UP	User program space
3	??	Undefined
4	??	Undefined
5	SD	Supervisor data space
6	SP	Supervisor program space
7	CS	CPU space

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:

```
cacr:0=..
     ↑  ↑↑
     |  |E (or .) = Cache enabled (disabled)
     | F (or .) = Cache frozen (unfrozen)
    Value of cacr
```

68030:

```
          Data  Instruction
         ┌──┴─┐ ┌┴─┐
cacr:3011=.-...-...
    ↑    ↑ ↑↑↑ ↑↑↑
    |    | ||E ||E (or .) = Cache enabled (disabled)
    |    | |F  |F (or .) = Cache frozen (unfrozen)
    |    | B   B (or .) = Burst fill enabled (disabled)
    |    W (or .) = Write allocate mode on (off)
    Value of cacr
```

68040:

```
cacr:80000000=(DE/ID)
     ↑   ↑
     |   |
     |   Instruction cache enabled (disabled)
     Data cache enabled (disabled)
```

68060:

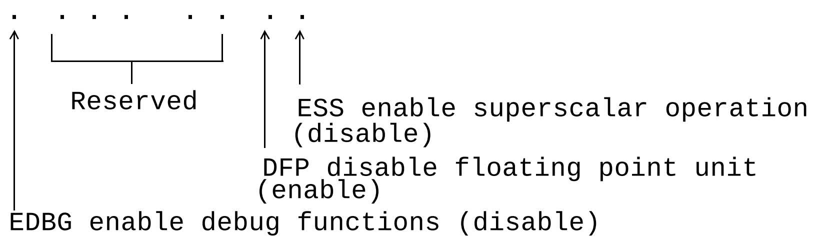
```
                     Data  Branch Instruction
                    Cache  Cache   Cache
                    ┌─┴─┐  ┌─┴─┐  ┌─┴─┐
cacr:A0808000=(E.S..---E..-----E..-------------)
             ↑      ↑↑↑↑↑  ↑↑↑   ↑↑↑
             |      |||||  |||   |||
             |      |||||  |||   ||H = 1/2 cache mode (full)
             |      |||||  |||   |A = No allocate mode(allocate)
             |      |||||  |||   E = Enable instruction cache
             |      |||||  |||          (disable)
   Value of cacr    |||||  ||U = Clear user branch cache
                    |||||  |C = Clear all branch cache
                    |||||  E = Enable branch cache (disable)
                    |||||
                    ||||H = 1/2 cache mode (full mode)
                    |||P = Disable Cpush invalidation (enable)
                    ||S = Enable store buffer (disable)
                    |A = No allocate mode (allocate)
                    E = Enable data cache (disable)
```

Note

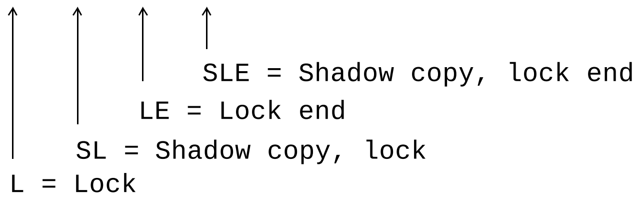
For the 68349, the CIC control registers are **not** displayed by the oa command. Use the memory examine commands to examine the CIC MCR Register.

For the 68060 processor, the `pcr` and `buscr` are interpreted as follows:

```
pcr = 04300101 (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 (. : . : . : . )
```



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) value, the status register (`sr`), and stack pointers (`msp`, `usp`, **and** `isp`).

The status register value is interpreted as follows:

```
sr:0000 (-----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 0001
pc:0010E3DC sr:2000 (--UI-0-----)t:OFF isp:0001CBEE msp:C001A4FC   ^us
```

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:0001DD98   ^is
```

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:0001DD98 isp:001CBEE    ^ms
```

Note

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:

```
fp0:40010000 D5555555 55555200 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.

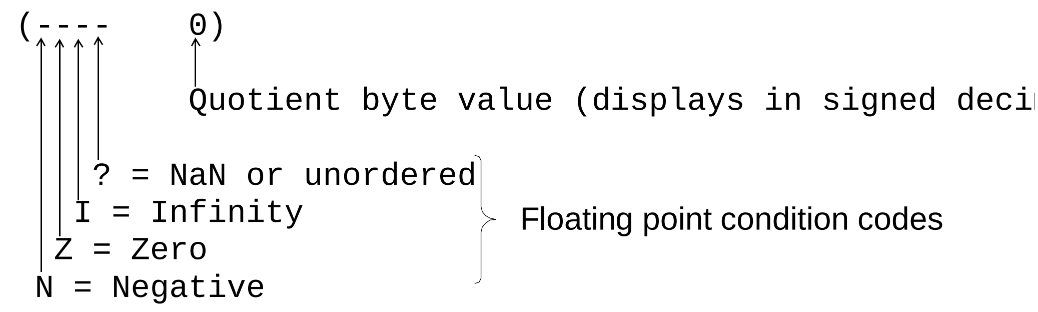
The floating point status registers display to the far right of the display:

`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 `fpsr` condition code and quotient bytes are displayed as follows:


(----   0)
Quotient byte value (displays in signed deci
? = NaN or unordered
I = Infinity
Z = Zero
N = Negative
Floating point condition codes

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:

`XE:(BSUN,SNAN,OPERR,OVFL,UNFL,DZ,INEX2,INEX1)` fpcr exception enable
`AX:(IOP,OVFL,UNFL,DZ,INEX,???,???,???)` fpsr accrued exception
`XS:(BSUN,SNAN,OPERR,OVFL,UNFL,DZ,INEX2,INEX1)` fpsr exception status

A full register display example follows:

```
dn:00000000 00000000 00000001 00000003 00000000 000000A2 00001050 0000
an:000152D2 00000000 00015F4E 00015F46 00015F42 00000000 0001CF30 0001
pc:00139364  cc: 04 (--Z--)
fp0:40010000 C0000000 00000000 fp4:7FFF0000 FFFFFFFF FFFFFFFF  fpcr: 0
fp1:7FFF0000 FFFFFFFF FFFFFFFF fp5:7FFF0000 FFFFFFFF FFFFFFFF fpiar: 0
fp2:7FFF0000 FFFFFFFF FFFFFFFF fp6:7FFF0000 FFFFFFFF FFFFFFFF  fpsr: 0
fp3:7FFF0000 FFFFFFFF FFFFFFFF fp7:7FFF0000 FFFFFFFF FFFFFFFF (----
AX:(INEX)
_exit+0x6           >DEADDEAD        add.l -8531(a5),d7
```

Note

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

	State/Processor			
Register	All / User	68000/10/60 /70/CPU32 Supervisor	68020/30/40 Supervisor-IRQ	68020/30/40 Supervisor-Master
`.a7`	`a7/usp`	`a7/ssp`	`isp`	`msp`
`.sp`	`a7/usp`	`a7/ssp`	`isp`	`msp`
`.ssp`[1]	`ssp`	`a7/ssp`	`isp`	`msp`
`.usp`	`a7/usp`	`a7/usp`	`usp`	`usp`
`.isp`[2]	`isp`	-	`isp`	`isp`
`.msp`[2]	`msp`	-	`msp`	`msp`

[1]Current supervisor stack pointer - all states

[2]User 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:

68030 MMU display

```
srp:5E82AE5FEFCFFFCF  ULim=EFCF DT=8 Local TA=EFCFFFCF
crp:72226000FDDF7EFF  ULim=FDDF DT=I Local TA=FDDF7EFF
 tc:03FFFFFF   PS=32K IS=F TIA=F TIB=F TIC=F TID=F (------SF)
tt0:FFFF0777           LBA=FF LAM=FF FCB=7 FCM=7 (-----IRM)
tt1:00000000           LBA=00 LAM=00 FCB=0 FCM=0 (-----CW-)
psr:    EE47 (BLS-WIM--T---7)
```

68040 MMU display

```
srp: 00000000  urp: 003F5E00  tc: 8000 (Enabled/4K)
dtt0: E01FA040 - lab=E0  lam=1F  attr=A040 (E01---00-10-----)
dtt1: 00FFA000 - lab=00  lam=FF  attr=A000 (E01---00-00-----)
itt0: 00000000 - lab=00  lam=00  attr=0000 (-00---00-00-----)
itt1: 00FFA000 - lab=00  lam=FF  attr=A000 (E01---00-00-----)
mmusr: 003E7031 - pa=003E7  (--00-01M---R)
```

68060 MMU display

```
urp:00000000    srp:00000000    tc:8000 DC=WT IC=WT (E---HH---)
dtt0:E01FA040 - lab=E0 lam=1F attr=A040 (E01---00-10-----)
dtt1:00FFA000 - lab=00 lam=FF attr=A000 (E01---00-00-----)
itt0:000000FF - lab=00 lam=00 attr=00FF (-00---00-11--W--)
itt1:00FFA000 - lab=00 lam=FF attr=A000 (E01---00-00-----)
```

68851 MMU display

```
srp:7FFF000100000000  ULim=0000 DT=P Local TA=00000000
crp:00080003003F8DD0  ULim=003F DT=8 Local TA=003F8DD0
drp:80000000000040A1  LLim=0000 DT=I Local TA=000040A1
 tc:82C08444   PS=4KB IS=0 TIA=8 TIB=4 TIC=4 TID=4 (E-----S-)
cal: 00 val: 00 scc: 7B ac: 0003
psr:    0804 (----W--------4)
pcsr:   0007 (-------------7)
bac:00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000
bad:00000212 00008000 08001010 10001200 280001C1 020000A0 00004001 004000
```

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.

Note

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

Examples

```
dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc:000F8FA8  cc: 04 (--Z--)
_exit+0x6           >DEADDEAD         add.l -8531(a5),d7
RomBug: .d4 100                                         set D4 to 100
RomBug: .                                               display registers
dn:00000000 00000000 00000001 00000003 00000100 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc:000F8FA8  cc: 04 (--Z--)
RomBug: .d4 .d2+.a6                                     set D4 using an expression
RomBug: .                                               display registers
dn:00000000 00000000 00000001 00000003 00025611 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc:000F8FA8  cc: 04 (--Z--)
```

```
RomBug: .fp0 4 set FP0 to 4.0
RomBug: .
dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc:000F8FA8  cc: 04 (--Z--)
fp0:40010000 80000000 00000000 fp4:7FFF0000 FFFFFFFF FFFFFFFF  fpcr: 0000 XN
fp1:7FFF0000 FFFFFFFF FFFFFFFF fp5:7FFF0000 FFFFFFFF FFFFFFFF fpiar: 00000000
fp2:7FFF0000 FFFFFFFF FFFFFFFF fp6:7FFF0000 FFFFFFFF FFFFFFFF  fpsr: 00000208
fp3:7FFF0000 FFFFFFFF FFFFFFFF fp7:7FFF0000 FFFFFFFF FFFFFFFF (----    0)
AX:(INEX) XS:(INEX2)
_exit+0x6           >DEADDEAD         add.l -8531(a5),d7
RomBug: .fp0 0x4                                set FP0 with hex value (left justified)
RomBug: .
dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc:000F8FA8  cc: 04 (--Z--)
fp0:40000000 00000000 00000000 fp4:7FFF0000 FFFFFFFF FFFFFFFF  fpcr: 0000 XN
fp1:7FFF0000 FFFFFFFF FFFFFFFF fp5:7FFF0000 FFFFFFFF FFFFFFFF fpiar: 00000000
fp2:7FFF0000 FFFFFFFF FFFFFFFF fp6:7FFF0000 FFFFFFFF FFFFFFFF  fpsr: 00000208
fp3:7FFF0000 FFFFFFFF FFFFFFFF fp7:7FFF0000 FFFFFFFF FFFFFFFF (----    0)
AX:(INEX) XS:(INEX2)
_exit+0x6           >DEADDEAD         add.l -8531(a5),d7
```

```
dis: .fp0 0x40200000aef5                                   another hex value
RomBug: .
dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc 000F8FA8  cc: 04 (--Z--)
fp0:40200000 AEF50000 00000000 fp4:7FFF0000 FFFFFFFF FFFFFFFF  fpcr: 0000 XN
fp1:7FFF0000 FFFFFFFF FFFFFFFF fp5:7FFF0000 FFFFFFFF FFFFFFFF fpiar: 00000000
fp2:7FFF0000 FFFFFFFF FFFFFFFF fp6:7FFF0000 FFFFFFFF FFFFFFFF  fpsr: 00000208
fp3:7FFF0000 FFFFFFFF FFFFFFFF fp7:7FFF0000 FFFFFFFF FFFFFFFF (----    0)
AX:(INEX) XS:(INEX2)
_exit+0x6           >DEADDEAD         add.l -8531(a5),d7
RomBug: dx &.fp0                                        display in decimal
[reg] - 40200000AEF5000000000000 11741167616.
```

```
RomBug: od                                 change register display format to decimal
RomBug: .fp7 4.2e10                        set FP7 to 42000000000.0
RomBug: .
dn:00000000 00000000 00000001 00000003 00000000 000000A6 00001210 00000000
an:0001DB78 00000000 0001E7EE 0001E7E2 0001E7DE 00000000 00025610 0001E770
pc:000F4FA8  cc: 04 (--Z--)
fp0:6.666666666666666         fp4:<NaN>                     fpcr: 0000 XN
fp1:<NaN>                     fp5:<NaN>                    fpiar: 00000000
fp2:<NaN>                     fp6:<NaN>                     fpsr: 00000208
fp3:<NaN>                     fp7:42000000000.00001         (----    0)
```

Change Machine Registers

Note

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

Examples

```
RomBug: .d4 100
RomBug: .
dn:00000000 00002700 00000002 00020000 00000100 00000001 FFFFE000 00004C00
an 00015100 FFFE2800 00010000 00015100 00005C00 000150F8 00010000 000150E4
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: .d4 .d2+.a6
RomBug: .
dn:00000000 00002700 00000002 00020000 00010002 00000001 FFFFE000 00004C00
an:00015100 FFFE2800 00010000 00015100 00005C00 000150F8 00010000 000150E4
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 FFFE2800 00010000 00015100 00005C00 000150F8 00010000 000150E4
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 FFFE2800 00010000 00015100 00005C00 000150F8 00010000 000150E4
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 FFFE2800 00010000 00015100 00005C00 000150F8 00010000 000150E4
pc:003C51A4 sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00000000 ^isp^
fp0:40000000 C90FCF80 DC337000 fp4:7FFF0000 FFFFFFFF FFFFFFFF fpcr: 0000 XN
fp1:40000000 C92FEF80 FC337000 fp5:7FFF0000 FFFFFFFF FFFFFFFF fpiar: 00000000
fp2:7FFF0000 FFFFFFFF FFFFFFFF fp6:7FFF0000 FFFFFFFF FFFFFFFF fpsr: 00000000
fp3:7FFF0000 FFFFFFFF FFFFFFFF fp7:7FFF0000 FFFFFFFF FFFFFFFF (----    0)
null_restore        >F36E050C         frestore fp_cir+$2(a6)
RomBug: od
Input radix = 16
Ram (hard) breakpoints
MPU type = 68030/68881
Show 68881 registers ON in decimal
Show MMU regs OFF
Show all cpu regs OFF
RomBug: .
dn:00000000 00002700 00000002 00020000 00010002 00000001 FFFFE000 00004C00
an:00015100 FFFE2800 00010000 00015100 00005C00 000150F8 00010000 000150E4
pc:003C51A4 sr:2700 (--SI-7-----)t:OFF msp:EC57A71C usp:00000000  ^isp^
fp0:3.14159                   fp4:<NaN>                      fpcr: 0000 XN
fp1:3.143550754510946         fp5:<NaN>                     fpiar: 00000000
fp2:<NaN>                     fp6:<NaN>                      fpsr: 00000000
fp3:<NaN>                     fp7:<NaN>                      (----    0)
null_restore        >F36E050C         frestore fp_cir+$2(a6)
```

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
`Scc`	Set according to condition
`TRAPcc`	Trap on condition
`FBcc`	Branch on floating condition
`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        >257C000010000088  move.l #$1000,$88(a2)
initspu+$18        >42AE0000          clr.l $0(a6)
initspu+$1C        >7208              moveq.l #$8,d1
initspu+$1E        >41EE000A          lea.l $A(a6),a0
initspu+$22        >2008              move.l a0,d0
initspu+$24        >61FF00000968      bsr.l clearmem
initspu+$2A        >303C7FFF          move.w #$7FFF,d0
initspu+$2E        >EFEE004F000A      bfins d0,$A(a6){$1:$F}
initspu+$34        >08AE0001000C      bclr.b #$1,$C(a6)
```

```
dis: di Main 5      disassemble 5 instructions
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
```

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    - 40000000C90FDAA22168C235 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 00002004 000DACB8 000DABBA 00000100 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^
0x0000714A   >202C02E0         move.l 736(a4),d0
RomBug: oa   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 00002004 000DACB8 000DABBA 00000100 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^
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 1 see we're not foolin'
0x0000000A    - 1E38FFF0 1E3EFFF0 1E440000 041E0000  .8.p.>.p.D......
RomBug: ov    display currently monitored vectors
 31  7C Level 7 Interrupt Autovector     stop on supervisor/user state
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>
  8  20 Privilege Violation              <not monitored>
  9  24 Trace                            <not monitored>
 10  28 A-Line                           <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 3FC User defined 255                 display on supervisor state
RomBug: ovud254  monitor and displayuser 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:
```

Chapter 4: Pentium and 80x86 Processors

The following is provided in this section for the Pentium and 80x86 processors:

-o Options

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

Table 4-1 80x86 Options

Option	Description
.	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

Table 4-1 80x86 Options (continued)

Option	Description
`p[<M>]<port_num>`	Change I/O port contents at `<port_num>` `<M>` Change mode `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 and display `<num>` Store new value and 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

Commands

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

Table 4-2 Commands

Command	Description
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 4-3 Registers

Register	Description
`.eax`	General purpose register
`.ebx`	General purpose register
`.ecx`	General purpose register
`.edx`	General purpose register
`.esi`	General purpose register
`.edi`	General purpose register
`.ebp`	General purpose register
`.esp`	Stack pointer
`.eip`	Instruction pointer register
`.eflags`	Status register
`.cs`	Code segment register
`.ds`	Data segment register
`.es`	Data segment register
`.fs`	Data segment register

Table 4-3 Registers (continued)

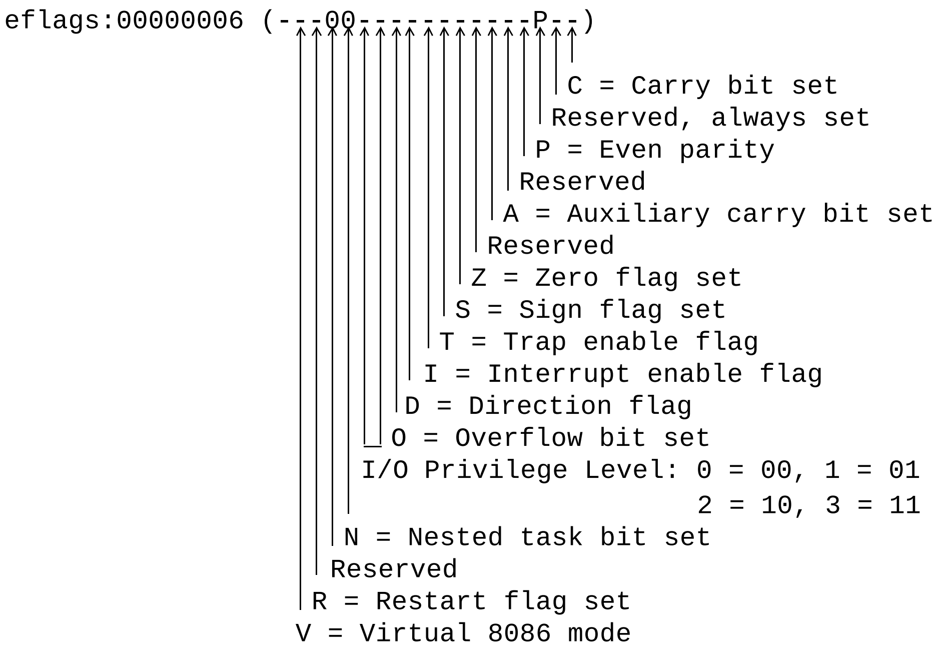
Register	Description
`.gs`	Data segment register
`.ss`	Stack segment register
`.gdtr`	Base register for global descriptor table
`.idtr`	Base register for interrupt descriptor table
`.ldtr`	Selector register for local descriptor table
`.tr`	Task State Segment (TSS) register
`.esp0 - .esp2`	Privilege level 0-2 registers
`.ss0 - .ss2`	Stack pointer for privilege level 0-2
`.dr0 -.dr3`	Debug registers
`.dr6`	Debug registers
`.dr7`	Debug registers
`.cr0 - .cr3`	Control registers
`.fcr`	Floating point control register
`.ftw`	Floating point tag register
`.fsr`	Floating point status register
`.st0 - .st7`	Floating point registers

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
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
eip: 0000B72E   eflags: 00000006 (---00-----------P--)
0x0000B72E   >8B5128           mov.l $28(%ecx),%edx
```

The `eflags` status control register value is interpreted as follows:



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

```
G-------------------------XTEMP
|                         |||||
|                         ||||Protection Enable
|                         |||Monitor Coprocessor
|                         ||Emulate Coprocessor
|                         |Task Switched
|                         Precessor Extension Type
Paging Enable
```

The `cr0` register value for the 80486 is interpreted as follows:

```
GCN----------A-W----------NXTEMP
|||          | |          ||||||
|||          | |          |||||Protection Enable
|||          | |          ||||Monitor Coprocessor
|||          | |          |||Emulate Coprocessor
|||          | |          ||Task Switched
|||          | |          |Processor Extension Type
|||          | |          Numerics Exception
|||          | Write Protect
|||          Alignment mask
||No write-through
|Cache Disable
Paging Enable
```

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:

```
eax: 00103F40   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
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---------)
0x0000B72E   >8B5128           mov.l $28(%ecx),%edx
```

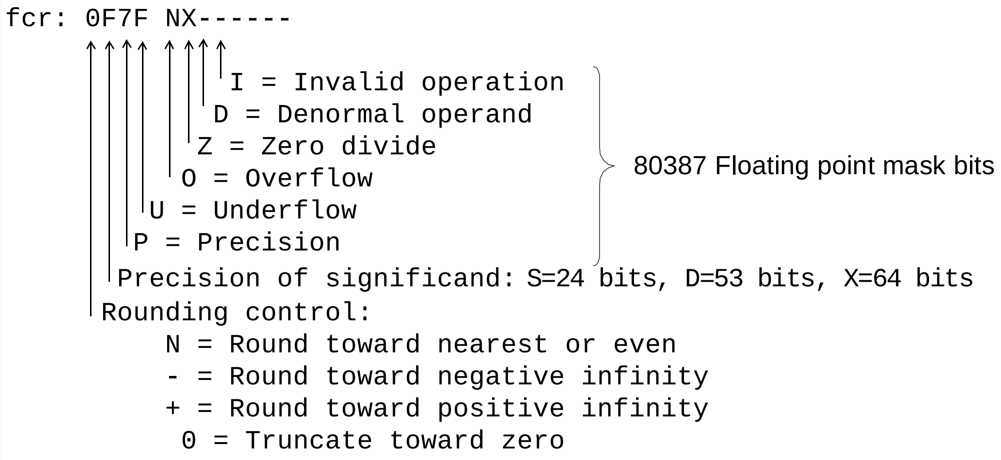
If the od command is used, the registers are in the following decimal format:

```
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   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
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---------)
0xB72E              >8B5128          mov.l $28(%ecx),%edx
RomBug:
```

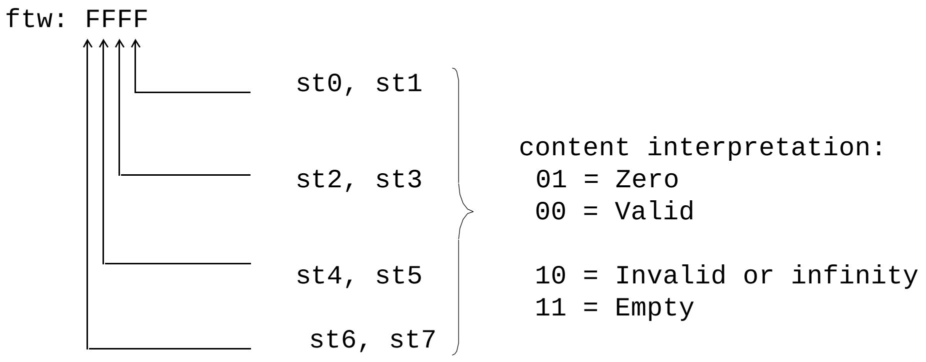
The floating point status registers display the following:

`fcr:0F7F ZXPUOZDI`	floating point control register
`ftw:FFFF`	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:



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 (CO).
 ||||||        Maps to "CF" (carry flag) in eflag reg.
 ||||| 1 = Floating point condition code (C1)
 ||||  P = Floating point condition code (c2).
 ||___      Maps to "PF" (parity flag) in eflags reg.
 ||  Indicates top of stack register: 000 = st0, 001 = st1, 010 = st2, ...
 | Z = Floating point condition code (C3). "ZF" (zeroflag) in eflags reg.

 B = Exception summary status (8087 compatibility)
```

A full register display example follows:

```
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
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
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---------)
0x0000B72E   >8B5128           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: .
eax: 0021F468   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
eip: 0000B72E   eflags: 00000002 (---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---------)
0xB72E              >8B5128          mov.l $28(%ecx),%edx
RomBug: .eax 100
RomBug: .
eax: 00000100   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
eip: 0000B72E   eflags: 00000002 (---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---------)
0xB72E              >8B5128          mov.l $28(%ecx),%edx
RomBug: .eax .ecx+.ebp
RomBug: .
eax: 0021F468   ebx: 00100000   ecx: 001018C4   edx: 00103F40
esi: 0010188C   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
eip: 0000B72E   eflags: 00000002 (---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---------)
xB72E              >8B5128          mov.l $28(%ecx),%edx
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   edi: 00101858   ebp: 0011DBA4   esp: 0011DB90
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
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---------)
0xB72E              >8B5128          mov.l $28(%ecx),%edx
RomBug:
```

Instruction Disassembly Memory Display

In the instruction disassembly display format, the `Jcc` 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 the case the processor performs.

Example

```
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            >8DB3061E0000    lea.l $1e06(%ebx),%esi
main+$F              >89F0          mov %esi,%eax
main+$11             >E8C7B20000     call  setjmp
main+$16            >8945F4        mov.l %eax,ss:$f4(%ebp)
main+$19             >3D00000000     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            >E8B5E8FFFF     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            >E8F1EBFFFF     call  get_vectors

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            >8DB3061E0000    lea.l $1e06(%ebx),%esi
main+$F              >89F0          mov %esi,%eax
```

Floating Point Memory Displays

The following is an example of floating point memory displays:

```
trace: d1 120400
$00120400    - DA0F4940 65657266 182D4454 FB210940  Z.I@eerf.-DT{!.@
dis: df 120400
$00120400    - 40490FDA 3.141592502593994
dis: dd 120408
$00120408    - 400921FB54442D18 3.141592653589793
dis: dx 120410
$00120410    - 4000 C90FDAA22168C234 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
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
                               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
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
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
cs: 0020  ds: 0038  es: 0038  fs: 0008  gs: 0038   ss: 0038
eip: 0000B72E   eflags: 00000002 (---00-------------)
0xB72E              >8B5128           mov.l $28(%ecx),%edx
RomBug: d 10 1
0x10                - C83800F0 54FF00F0 745700F0 C83800F0
H8.pT..ptW.pH8.p
dis: ov
RomBug: ov 10
10  40 Coprocessor error               stop on supervisor/user state
RomBug: ov 4 7
 4  10 Overflow                        stop on supervisor/user state
 5  14 Bounds check                    stop on supervisor/user state
 6  18 Invalid opcode                  stop on supervisor/user state
 7  1C Coprocessor not available       stop on supervisor/user state
RomBug: ov
 4  10 Overflow                        stop on supervisor/user state
 5  14 Bounds check                    stop on supervisor/user state
 6  18 Invalid opcode                  stop on supervisor/user state
 7  1C Coprocessor not available       stop on supervisor/user state
10  40 Coprocessor error               stop on supervisor/user state
RomBug: ov-4 10
 4  10 Overflow                        <not monitored>
 5  14 Bounds check                    <not monitored>
 6  18 Invalid opcode                  <not monitored>
 7  1C Coprocessor not available       <not monitored>
 8  20 Double fault                    <not monitored>
 9  24 Coprocessor segment overrun     <not monitored>
 A  28 Invalid TSS                     <not monitored>
 B  2C Segment not present             <not monitored>
 C  30 Stack exception                    <not monitored>
 D  34 General protection                 <not monitored>
 E  38 Page fault                         <not monitored>
 F  3C Unassigned/Reserved                <not monitored>
10  40 Coprocessor error                  <not monitored>
RomBug: ov
RomBug: ovsd40
40 100 User defined $40                   display on supervisor state
RomBug: ovud41
41 104 User defined $41                   display on user state
RomBug: ov
40 100 User defined $40                   display on supervisor state
41 104 User defined $41                   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
RomBug:
```

Chapter 5: PowerPC Processors

This chapter discusses PowerPC processore. The suppored 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

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

Table 5-1 PowerPC Options

Option	Description
`.`	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
`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 `r` read access `w` write access `rw` read/write access (default)

Table 5-1 PowerPC Options

Option	Description
`wd`	Show data watch point
`wi<addr>`	Set instruction watch point for `addr`
`w{i}`	Show instruction watch point

Commands

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

Table 5-2 Commands

Command	Description
e	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

Supported Registers

The following tables define user and supervisor registers specific to each processor. Table 5-3 identifies user registers common to the supported processors (exceptions are noted in the table's legend). Table 5-4 identifies user registers unique to the 601 processor. Table 5-5 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

Register Name	Alias Register Name	Description
`r0 - r31 †`	`gpr0 - gpr31`	General purpose registers
`cr`	`-`	Condition register
`f0 - f31 *(1)`	`fpr0 - fpr31`	Floating point registers
`fpscr *(1)`	`-`	Floating point status and control register
`xer`	`spr1`	Integer exception register
`lr`	`spr8`	Link register
`pc`	`-`	Program counter

Table 5-3 PowerPC User Registers (continued)

Register Name	Alias Register Name	Description
`msr`	`-`	Machine status register
`sp`	`gpr1 *(2)`	Stack pointer
`ctr`	`spr9`	Count register
`tbl *(3)`	`spr268`	Time base lower (read only)
`tbu *(3)`	`spr269`	Time base upper (read only)

† 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 5-4 601-Specific User Registers

Register Name	Alias Register Name	Description
`mq`	`spr0`	MQ register
`rtcu`	`spr4`	RTC upper register (read only)
`rtcl`	`spr5`	RTC lower register (read only)

Supervisor Registers

Table 5-5 Supervisor Registers Sorted Numerically

SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
0			mq				
1	xer	xer	xer	xer	xer	xer	xer
8	lr	lr	lr	lr	lr	lr	lr
9	ctr	ctr	ctr	ctr	ctr	ctr	ctr
18		dsisr	dsisr	dsisr	dsisr	dsisr	dsisr
19		dar	dar	dar	dar	dar	dar
20			rtcu (write only)				
21			rtcl (write only)				
22	dec	dec	dec	dec	dec	dec	dec
25			sdr1	sdr1	sdr1	sdr1	
26	srr0	srr0	srr0	srr0	srr0	srr0	srr0
27	srr1	srr1	srr1	srr1	srr1	srr1	srr1
80		eie					eie
81		eid					eid

Table 5-5 Supervisor Registers Sorted Numerically (continued)

SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
82		nri					nri
144		cmpa					cmpa
145		cmpb					cmpb
146		cmpc					cmpc
147		cmpd					cmpd
148		ecr					icr
149		der					der
150		counta					counta
151		countb					countb
152		cmpe					cmpe
153		cmpf					cmpf
154		cmpg					cmpg
155		cmph					cmph
156		lctrl1					lctrl1
157		lctrl2					lctrl2
158		ictrl					ictrl
159		bar					bar
272	sprg0	sprg0	sprg0	sprg0	sprg0	sprg0	sprg0

Table 5-5 Supervisor Registers Sorted Numerically (continued)

SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
273	sprg1	sprg1	sprg1	sprg1	sprg1	sprg1	sprg1
274	sprg2	sprg2	sprg2	sprg2	sprg2	sprg2	sprg2
275	sprg3	sprg3	sprg3	sprg3	sprg3	sprg3	sprg3
282			ear	ear	ear	ear	
284		tbl		tbl	tbl	tbl	
285		tbu		tbu	tbu	tbu	
287	pvr	pvr	pvr		pvr		pvr
528			ibat0u	ibat0u	ibat0u	ibat0u	
529			ibat0l	ibat0l	ibat0l	ibat0l	
530			ibat1u	ibat1u	ibat1u	ibat1u	
531			ibat1l	ibat1l	ibat1l	ibat1l	
532			ibat2u	ibat2u	ibat2u	ibat2u	
533			ibat2l	ibat2l	ibat2l	ibat2l	
534			ibat3u	ibat3u	ibat3u	ibat3u	
535			ibat3l	ibat3l	ibat3l	ibat3l	
536				dbat0u	dbat0u	dbat0u	
537				dbat0l	dbat0l	dbat0l	
538				dbat1u	dbat1u	dbat1u	

Table 5-5 Supervisor Registers Sorted Numerically (continued)

SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
539				dbat1l	dbat1l	dbat1l	
540				dbat2u	dbat2u	dbat2u	
541				dbat2l	dbat2l	dbat2l	
542				dbat3u	dbat3u	dbat3u	
543				dbat3l	dbat3l	dbat3l	
560		ic_cst					ic_cst
561		icadr					ic_adr
562		icdat					ic_dat
568							dc_cst
569							dc_adr
570							dc_dat
630		dpdr					dpdr
631							dpir
638							immr
784							mi_ctr
786							mi_ap
787							mi_epn
789							mi_mi_twc

Table 5-5 Supervisor Registers Sorted Numerically (continued)

SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
790							`mi_rpn`
792							`md_ctr`
793							`m_casid`
794							`md_ap`
795							`md_epn`
796							`m_twb`
797							`md_twc`
798							`md_rpn`
799							`m_tw`
816							`mi_dbcam`
817							`mi_dbram0`
818							`mi_dbram1`
824							`md_dbcam`
825							`md_dbram0`
826							`md_dbram1`

Table 5-5 Supervisor Registers Sorted Numerically (continued)

SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
952						mmcr0	
953						pmc1	
954						pmc2	
955						sia	
959						sda	
976				dmiss	dmiss		
977				dcmp	dcmp		
978				hash1	hash1		
979				hash2	hash2		
980	esr			imiss	imiss		
981	dear			icmp	icmp		
982	evpr			rpa	rpa		
984	tsr			tcr			
986	tcr			ibr			
987	pit			esasrr			
988	tbhi						
989	tblo						
990	srr2			sebr			

Table 5-5 Supervisor Registers Sorted Numerically (continued)

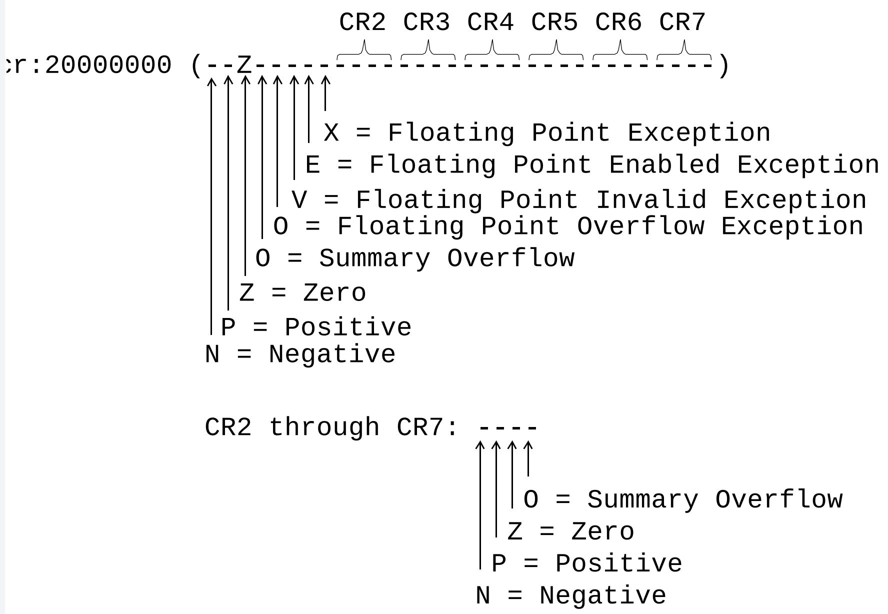
SPR #	403 SPR Name	505 SPR Name	601 SPR Name	602 SPR Name	82xx/ 603 SPR Name	604 SPR Name	8xx SPR Name
991	srr3			ser			
1008	dbsr		hid0	hid0	hid0		
1009			hid1	hid1			
1010	dbcr		iabr	iabr	iabr	iabr	
1012	iac1						
1013	iac2		dabr	dabr		dabr	
1014	dac1						
1015	dac2						
1018	dccr						
1019	iccr						
1020	pbl1						
1021	pbu1			sp			
1022	pbl2	fpecr		lt			
1023	pbu2		pir			pir	

Display Information

Normal register display:

```
 r0:00FE8A60 00FE8A00 00029C40 00000000 00FE8A08 00000003 00004ED8 001
 r8:00105B90 00FE8DB8 00000003 000000DC 00000000 0020F3D0 0002B4B8 000
r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 000
r24:00000000 00000700 00000001 00000001 00FE8150 0024978C 00000007 000
pc:0020F3D0 lr: 48003614 ctr: 00004400 msr: 00001030 (---C------ID----
cr:20000000 (--Z----------------------------)
0x20F3D0            >7C0802A6        mflr r0
```

The `cr` value is interpreted as follows:



CR2 CR3 CR4 CR5 CR6 CR7
cr: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:00001030 (---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:  8000018
hash1: 00FC9900 hash2: 00FD66C0 rpa:   0023F199 sdr1:  00FC000
sr0:   20000003 sr1:   20000003 sr2:   20000003 sr3:   2000000
sr4:   20000003 sr5:   20000003 sr6:   20000003 sr7:   2000000
sr8:   20000003 sr9:   20000003 sr10:  20000003 sr11:  2000000
sr12:  20000003 sr13:  20000003 sr14:  20000003 sr15:  2000000
ibat0u:000003FE ibat1u:800000FE ibat2u:00000000 ibat3u:0000000
ibat0l:0000001A ibat1l:8000003A ibat2l:00000000 ibat3l:0000000
dbat0u:000003FE dbat1u:800000FE dbat2u:00000000 dbat3u:0000000
dbat0l:0000001A dbat1l:8000003A dbat2l:00000000 dbat3l:0000000
```

The following register information displays following an oa command:

```
srr0:  00102144 srr1:  00003030
sprg0: 00029C40 sprg1: 00FE8144 sprg2: 40000000 sprg3: 00FE8A6
dsisr: 40000000 dar:   43A67C43 ear:   00000000 sdr1:  00FC0001
```

601/603 floating point register dump:

```
 f0: fff8000000000000 0000000000000000 0000000000000000 00000000000000
 f4: 0000000000000000 0000000000000000 0000000000000000 00000000000000
 f8: 0000000000000000 0000000000000000 0000000000000000 00000000000000
f12: 0000000000000000 0000000000000000 0000000000000000 00000000000000
f16: 0000000000000000 0000000000000000 0000000000000000 00000000000000
f20: 0000000000000000 0000000000000000 0000000000000000 00000000000000
f24: 0000000000000000 0000000000000000 0000000000000000 00000000000000
f28: 0000000000000000 0000000000000000 fff8000000000000 00000000000000
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 d 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    - 80028538 9003002C 8002B208 60030000  ...8...,..2.`...
$00100450    - 48000505 BBC10008 38210010 80010004  H...;A..8!......
$00100460    - 7C0803A6 4E800020 9421FFF8 8002B23C  |..&N.. .!.x..2<
$00100470    - 28000040 40820008 48000028 8002B23C  (..@@...H..(..2<
$00100480    - 30A00001 90A2B23C 38C00003 7C003030  0 ..."2<8@..|.00
$00100490    - 38A28020 7CA50214 7C80292E 90650004  8”. |%..|.)..e..
$001004A0    - 38210008 4E800020 7C0802A6 90010004  8!..N.. |..&....
$001004B0    - 9421FFE8 BFC10008 607F0000 609E0000  .!.h?A..`...`...
$001004C0    - 38000094 90010010 38610010 3882B208  8.......8a..8.2.
$001004D0    - 38A00000 480009AD 28030000 41820008  8 ..H..-(...A...
$001004E0    - 4BFFFC05 8082B208 80610010 38000000  K.|...2..a..8...
$001004F0    - 5463F0BE 4800001C 60850000 30840004  Tcp>H...`...0...d
```

Change Machine Registers

Examples

```
RomBug:  .
 r0:00000000 00FE80B8 00029C40 00000000 00000000 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 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 (----------------------------------)
0x0020BA2C   >7FE00124         mtmsr r31

RomBug: .r4 100
RomBug: .
 r0:00000000 00FE80B8 00029C40 00000000 00000100 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 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 (----------------------------------)
0x0020BA2C   >7FE00124         mtmsr r31
```

```
RomBug: .r4.r2+.r6
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 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 (----------------------------------)
0x0020BA2C   >7FE00124         mtmsr r31

RomBug: .f0 4
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 00009030
pc: 0020BA2C lr: 0020BA2C ctr: 00004400 msr: 00001030 (---C------ID----)
cr: 40000000 (-P------------------------------)
 f0: 4                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 (----------------------------------)
0x0020BA2C   >7FE00124         mtmsr r31
```

```
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 00009030
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 0x0000000c90fcf80dc337000
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 00009030
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: od
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            >60003AE0        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    - 40490FDB 3.141592741012573
dis: dd 20000
$00020000    - 400921FB54442D18 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
 r8:00286288 00000000 00000000 00000000 00000000 001CE3AC 00000000 00000000
r16 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24:00000000 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: 20000000 sprg3: 0028625C
dsisr: 00000000 dar:   00000000 ear:   00000000 sdr1:  00FC0001
 r0:00000001 0028625C 0028BFFC 00000001 00286298 002862A0 00284D9C 00000002
 r8:00286288 00000000 00000000 00000000 00000000 001CE3AC 00000000 00000000
r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24:00000000 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: 20000000 sprg3: 0028625C
dsisr: 00000000 dar:   00000000 ear:   00000000 sdr1:  00FC0001
 r0:00000001 0028625C 0028BFFC 00000001 00286298 002862A0 00284D9C 00000002
 r8:00286288 00000000 00000000 00000000 00000000 001CE3AC 00000000 00000000
r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 002862B0
pc: 001C66D4 lr: 001C65B0 ctr: 00000000 msr: 00009000 (X--C------------)
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  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
dsisr: 00000000 dar:   00000000 ear:   00000000 sdr1:  00FC0001
dmiss: 001C63C2 imiss: 001662C4 dcmp:  FE9EC100 icmp:  FE9EC100
hash1: 00FD3900 hash2: 00FCC6C0 rpa:   00166199 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 0028BFFC 00000001  00286298 002862A0 00284D9C 00000002
 r8:00286288 00000000 00000000 00000000 00000000 001CE3AC 00000000 00000000
r16:00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
r24:00000000 00000000 00000000 00000000 00000000 00000000 00000000 002862B0
pc: 001C66D4 lr: 001C65B0 ctr: 00000000 msr: 00009000 (X--C------------)
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 fff8000000000000 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: ovud 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>
```

Chapter 6: ARM Processor

The following information is provided in this section for the ARM processors:

- **-o 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 are supported by the ARM version of RomBug.

Table 6-1 Options

Option	Description
`.`	Display registers upon monitored exception
`a`	Toggle control register display
`d`	Toggle FP decimal register display
`f`	Toggle FP register display
`m`	Toggle MMU register display

Commands

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

Table 6-2 Commands

Command	Description
e	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

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.

Table 6-3 Operating Modes

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

†† = 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 6-4 General Purpose (gp) Registers

Register Name	Alias Register Name	Description
`r0`		Callee saved register for locals and temporaries
`r1`		Callee saved register for locals and temporaries
`r2`		Callee saved register for locals and temporaries
`r3`		Callee saved register for locals and temporaries
`r4`		Callee saved register for locals and temporaries
`r5`		Callee saved register for locals and temporaries
`r6`	`gp`	Global data pointer

Table 6-4 General Purpose (gp) Registers (continued)

Register Name	Alias Register Name	Description
r7		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
r8		Second required registrable argument. Caller saved register for locals and temporaries
r9		Third required registrable argument. Caller saved register for locals and temporaries
r10		Fourth required registrable argument. Caller saved register for locals and temporaries
r11		Caller saved register for locals and temporaries
r12	cp	Code constant pointer
r13	sp	Stack pointer
r14	lr	Link register
r15	pc	Program counter

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

Register Name	Alias Register Name	Description
`f0`		First required `fp` argument or the `fp` function return value (caller-saved)
`f1`		Second required `fp` argument (caller-saved)
`f2`		Third required `fp` argument (caller-saved)
`f3`		Fourth required `fp` argument (caller-saved)
`f4`		Callee-saved register
`f5`		Callee-saved register
`f6`		Callee-saved register
`f7`		Callee-saved register
`fpcr0`	`fpsr`	Floating point status register
`fpcr15`	`fpcr*`	Floating point control register

* 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)

Register Name	Alias Register Name	Description
`scc0`	`id`	Processor ID
`scc1`	`ctrl`	Control register
`scc2`	`ttbr`	Translation table base register
`scc3`	`dacr`	Domain access control register
`scc4*`	`fsr`	Fault status register
`scc5`	`far`	Fault address register
`scc7`	`cf`	Cache operations
`scc8*`	`tlbf`	Translation lookaside buffer (TLB) operations
`scc9 - scc15`	-	Reserved

* Level 4 only

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 and Table 6-8.

Table 6-7 ARM State General Purpose Registers

System = & User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6 (gp)	r6 (gp)	r6 (gp)	r6 (gp)	r6 (gp)	r6 (gp)
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12 (cp)	r12_fiq (cp)	r12 (cp)	r12 (cp)	r12 (cp)	r12 (cp)

Table 6-7 ARM State General Purpose Registers (continued)

System = & User	FIQ	Supervisor	Abort	IRQ	Undefined
r13 (sp)	r13_fiq (sp)	r13_svc (sp)	r13_abt (sp)	r13_irq (sp)	r13_und (sp)
r14 (lr)	r14_fiq (lr)	r14_svc (lr)	r14_abt (lr)	r14_irq (lr)	r14_und (lr)
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)

= Level 4 only

Table 6-8 Current Program Status Registers (cpsr)

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	spsr_fiq	spsr_svc	spsr_abt	spsr_irq	spsr_und

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 THUMB State General Purpose Registers and Table 6-10 Current Program Status Registers (cpsr).

Table 6-9 THUMB State General Purpose Registers

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6 (gp)	r6 (gp)	r6 (gp)	r6 (gp)	r6 (gp)	r6 (gp)
r7	r7	r7	r7	r7	r7
r13 (sp)	sp_fiq (sp)	sp_svc (sp)	sp_abt (sp)	sp_irq (sp)	sp_und (sp)
r14 (lr)	lr_fiq (lr)	lr_svc (lr)	lr_abt (lr)	lr_irq (lr)	lr_und (lr)
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)

Table 6-10 Current Program Status Registers (cpsr)

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	spsr_fiq	spsr_svc	spsr_abt	spsr_irq	spsr_und

Display Information

Register displays defined in the section are:

- Normal
- Program Status Register
- Floating Point Status Register

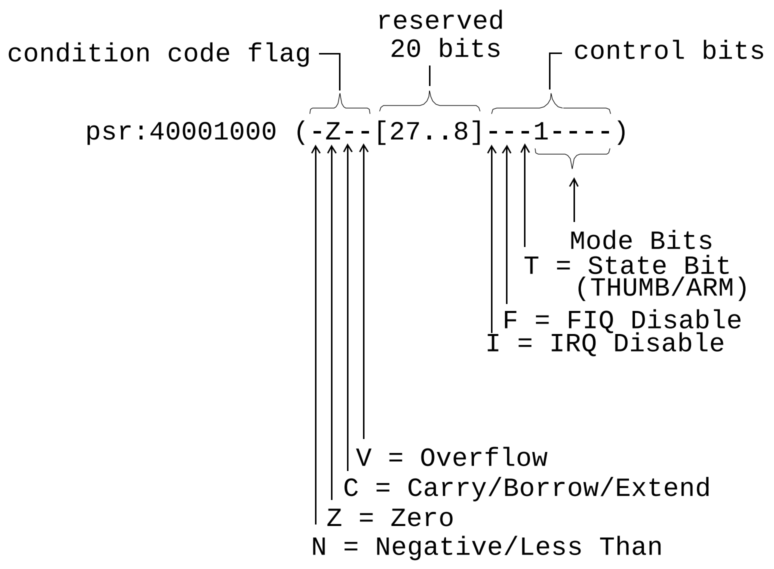
Normal Register

Normal register display:

```
00000000
```

Program Status Register

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



Mode bits values are defined in Table 6-11 PSR Mode Bit Values.

Table 6-11 PSR Mode Bit Values

Value	Mode
`10000`	User
`10001`	FIQ
`10010`	IRQ
`10011`	Supervisor
`10111`	Abort
`11011`	Undefined
`11111`	System =

= Level 4 only

Floating Point Status Register

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

```
               system control byte┐         ┌ cumulative
exception trap enable byte┐       │         │ exception
    system ID byte┐       │       │         │ flag byte
               ___^___ ___^___ ___^___ ___^___
 fpsr:40001000(--------------------------------)
```

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
RomBug: .
r0 : 001004BC r1 : 0010053C r2 : 00100480 r3 : 00000001 r4 : 00000100
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
RomBug: .
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)
0x00008DE4    >288090E5         ldr r8,[r0,#0x28]
```

Instruction Disassembly

```
RomBug: di Main
main          >0DB0A0E1          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
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: 20000093 (--C---------------------I--10011)
0x00008DE4   >288090E5         ldr r8,[r0,#0x28]
RomBug: oa
MPU type = 3, Input radix = 16
Ram (hard) breakpoints
Show control regs ON
Show MMU regs OFF
```

```
RomBug: .
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 : 00008DE4
pc: 00008DE4 cpsr: 20000093 (--C---------------------I--10011)
0x00008DE4   >288090E5         ldr r8,[r0,#0x28]

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

RomBug: .
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 : 00008DE4
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>
```

Chapter 7: SuperH Processors

The following is provided in this section for SuperH processors:

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

Supported processors are:

- SH7709
- SH7750

-o Options

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

Table 7-1 RomBug Options

Option	Description
`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
`x`	Toggle disassembly hex output format

Table 7-2 SH Options

Option	Description
`.`	Display registers upon monitored exception
`a`	Toggle control registers display

Table 7-2 SH Options

Option	Description
`d`	Toggle FP decimal register display
`f`	Toggle FP register display
`k [<watch_num>]`	Kill watch point(s) <watch_num>: watch point number If <watch_num> not present, kill all WPs
`m`	Toggle MMU registers display
`w`	Display table of active watch points/sequences
`w<cond>[#<cond> ...]`	Set a watch point or sequence per <cond>(s) (maximum number of <cond>s is eight); execution breaks when all <cond>s met in left-to-right order
`w?`	Above watch <cond> description & syntax help

Commands

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

Table 7-3 SuperH Commands

Command	Description
e	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

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

Register Name	Alias	Banked	Description
`r0`		`Y(.r0b)`	Integral return values. Caller saved register for locals and temporaries.
`r1`		`Y(.r1b)`	Caller saved register for locals and temporaries.
`r2`		`Y(.r2b)`	Caller saved register for locals and temporaries.
`r3`		`Y(.r3b)`	Caller saved register for locals and temporaries.
`r4`		`Y(.r4b)`	First required registrable function argument. First double return value. Caller saved register for locals and temporaries.
`r5`		`Y(.r5b)`	Second required registrable function argument. Second double return value. Caller saved register for locals and temporaries.

Table 7-4 SuperH General Purpose Registers (continued)

Register Name	Alias	Banked	Description
`r6`		`Y(.r6b)`	Third required registrable function argument. Caller saved register for locals and temporaries.
`r7`		`Y(.r7b)`	Fourth required registrable function argument.
`r8`		`N`	Callee saved register for locals and temporaries
`r9`		`N`	Callee saved register for locals and temporaries
`r10`		`N`	Callee saved register for locals and temporaries
`r11`		`N`	Callee saved register for locals and temporaries
`r12`		`N`	Callee saved register for locals and temporaries
`r13`	`cp`	`N`	Callee saved register for locals and temporaries
`r14`	`gp`	`N`	Global data pointer
`r15`	`sp`	`N`	Stack pointer

Control Registers

Table 7-5 SuperH Control Registers

Register Name	Description
`sr`	Status register
`gbr`	Global base register
`vbr`	Vector base register

System Registers

Table 7-6 SuperH System Registers

Register Name	Description
`mach`	Multiply and accumulate high register
`macl`	Multiply and accumulate low register
`pr`	Procedure register
`pc`	Program counter

Floating Point Registers (SH7750 only)

Table 7-7 SH-4 Floating Point Registers

Register Name	Alias	Banked	Description
`f0`		`Y(.x0)`	Floating point single precision register
`f1`		`Y(.x1)`	Floating point single precision register
`f2`		`Y(.x2)`	Floating point single precision register
`f3`		`Y(.x3)`	Floating point single precision register

Table 7-7 SH-4 Floating Point Registers

Register Name	Alias	Banked	Description
`f4`		`Y(.x4)`	Floating point single precision register
`f5`		`Y(.x5)`	Floating point single precision register
`f6`		`Y(.x6)`	Floating point single precision register
`f7`		`Y(.x7)`	Floating point single precision register
`f8`		`Y(.x8)`	Floating point single precision register
`f9`		`Y(.x9)`	Floating point single precision register
`f10`		`Y(.x10)`	Floating point single precision register
`f11`		`Y(.x11)`	Floating point single precision register
`f12`		`Y(.x12)`	Floating point single precision register
`f13`		`Y(.x13)`	Floating point single precision register
`f14`		`Y(.x14)`	Floating point single precision register

Table 7-7 SH-4 Floating Point Registers

Register Name	Alias	Banked	Description
`f15`		`Y(.x15)`	Floating point single precision register
`dr0`		`Y(.xd0)`	Floating point double precision register
`dr2`		`Y(.xd2)`	Floating point double precision register
`dr4`		`Y(.xd4)`	Floating point double precision register
`dr6`		`Y(.xd6)`	Floating point double precision register
`dr8`		`Y(.xd8)`	Floating point double precision register
`dr10`		`Y(.xd10)`	Floating point double precision register
`dr12`		`Y(.xd12)`	Floating point double precision register
`dr14`		`Y(.xd14)`	Floating point double precision register
`fpscr`		`N`	Floating point status and control register
`fpul`		`N`	Floating point communication register

Display Information

Register displays defined in this section are:

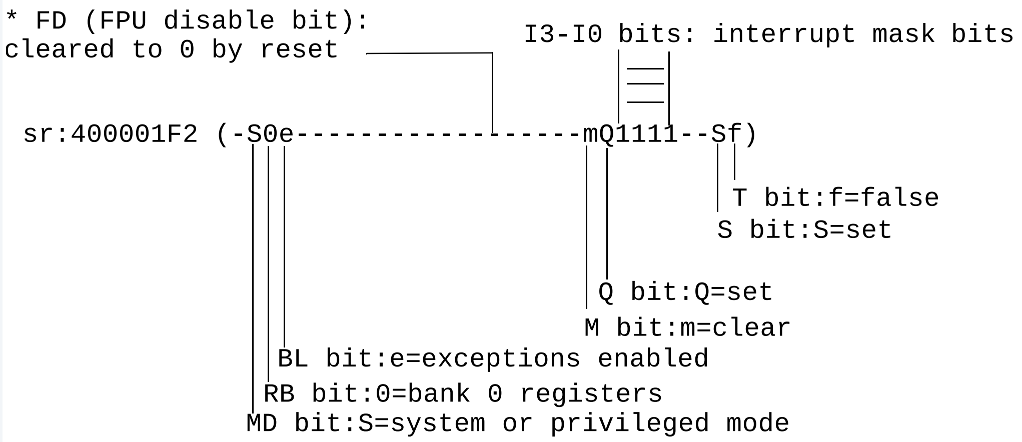
- Normal
- Status
- MMU Control
- Floating Point Status and Control

Normal Register Display

The normal register display for SuperH is:

```
00000000
```

Status Register Display



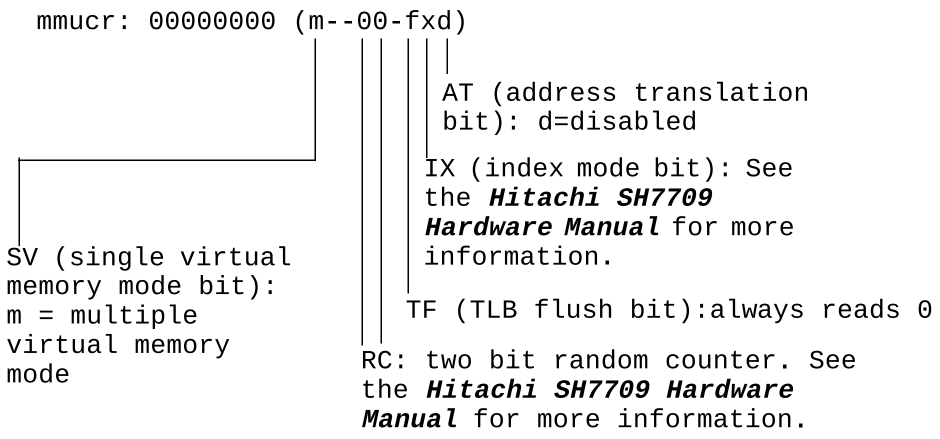
```
sr:400001F2 (-S0e-----------------mQ1111--Sf)
```

* FD (FPU disable bit):
cleared to 0 by reset
I3-I0 bits: interrupt mask bits
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 (SH7709)

Default value is 00000000.



```
mmucr: 00000000 (m--00-fxd)
```

AT (address translation bit): d=disabled

IX (index mode bit): See the ***Hitachi SH7709 Hardware Manual*** for more information.

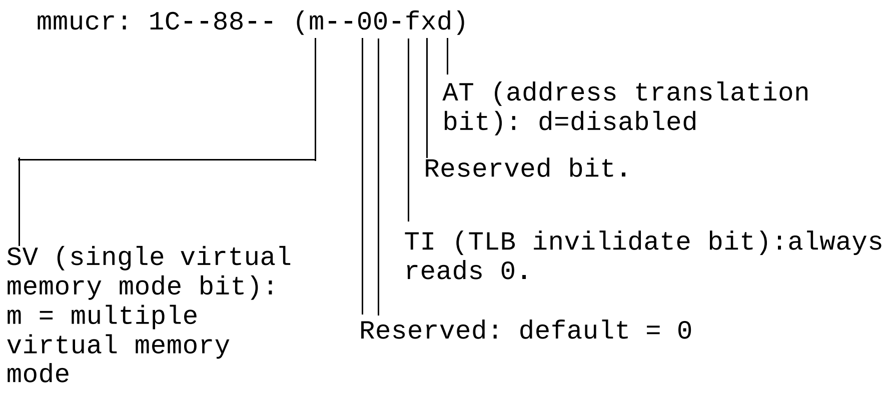
TF (TLB flush bit):always reads 0

RC: two bit random counter. See the ***Hitachi SH7709 Hardware Manual*** for more information.

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

MMU Control Register Display (SH7750)

Default value is 00000000.



```
mmucr: 1C--88-- (m--00-fxd)
```

AT (address translation bit): d=disabled

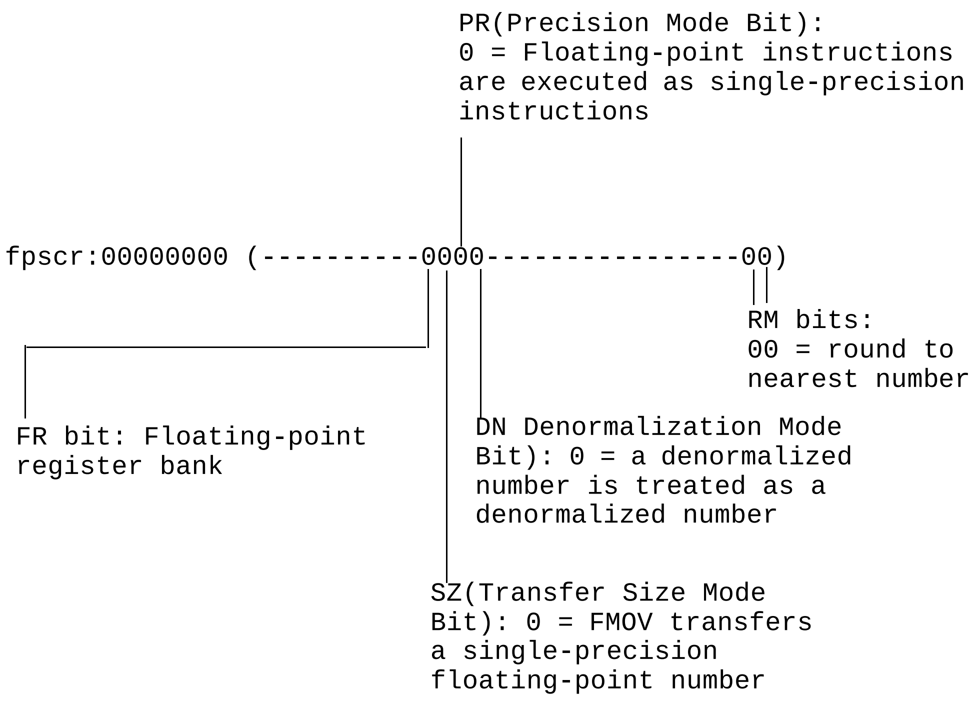
Reserved bit.

TI (TLB invilidate bit):always reads 0.

Reserved: default = 0

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

Floating Point Status and Control Register Display (SH7750 only)



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

```
fpscr:00000000 (----------0000---------------00)
```

FR bit: Floating-point register bank

DN Denormalization Mode Bit): 0 = a denormalized number is treated as a denormalized number

RM bits:
00 = round to nearest number

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

Change Machine Registers

Example

```
RomBug: .
r0: 00000788 8C0012D0 8C0012D0 9FFF8CB2  8C000E3C 8C000000 8C000E3C 800292B0
r8: 8C000E3C 8C000DF8 8C000EBC 00000001  00000000 E0007FFC 8C007FFC 8C023FC4
pr: 80000CE6 mach: FFFFFFFF macl: FFFFFFFF gbr: FFFFFFFF
pc: 80000CE6 sr: 400001F2 (-S0e------------------mQ1111--Sf)
0x80000CE6   >518A             mov.l @(0x28,r8),r1

RomBug: .r4 100
RomBug: .
r0: 00000788 8C0012D0 8C0012D0 9FFF8CB2  00000100 8C000000 8C000E3C 800292B0
r8: 8C000E3C 8C000DF8 8C000EBC 00000001  00000000 E0007FFC 8C007FFC 8C023FC4
pr: 80000CE6 mach: FFFFFFFF macl: FFFFFFFF gbr: FFFFFFFF
pc: 80000CE6 sr: 400001F2 (-S0e------------------mQ1111--Sf)
0x80000CE6   >518A             mov.l @(0x28,r8),r1

RomBug: .r4 .r2+.r6
RomBug: .
r0: 00000788 8C0012D0 8C0012D0 9FFF8CB2  1800210C 8C000000 8C000E3C 800292B0
r8: 8C000E3C 8C000DF8 8C000EBC 00000001  00000000 E0007FFC 8C007FFC 8C023FC4
pr: 80000CE6 mach: FFFFFFFF macl: FFFFFFFF gbr: FFFFFFFF
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 8C15EEC8 8C001360 0000000C 00000000 00000000 8FFDFFBC 00000000
r8: 40000001 00000000 0C15EFD0 00000000 00000000 00000000 8C008980 8FFDFF80
pr: 8C02E874 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C02E874 sr: 4000000F1 (-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
fpscr: 00000000 (----------0000---------------00)
r0: 00000A04 8C15EEC8 8C001360 0000000C 00000000 00000000 8FFDFFBC 00000000
r8: 40000001 00000000 0C15EFD0 00000000 00000000 00000000 8C008980 8FFDFF80
pr: 8C02E874 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C02E874 sr: 4000000F1 (-S0e------------------mq1111--sT)
0x8C02E874   >480E         ldc r8,sr

RomBug: dx &.f4
[reg] - 40490FD000000000 50.1235352
dis: .f4 0x0000000c90fcf80dc337000
RomBug: .
 f0: 4                   0                  0                  0
 f4: 0e-309              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                  0e-309             0
dr8: 0                   0                  0                  0
xd0: 0                   0                  0                  0
xd8: 0                   0                  0                  0
fpscr: 00000000 (----------0000---------------00)
r0: 00000A04 8C15EEC8 8C001360 0000000C 00000000 00000000 8FFDFFBC 00000000
r8: 40000001 00000000 0C15EFD0 00000000 00000000 00000000 8C008980 8FFDFF80
pr: 8C02E874 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C02E874 sr: 4000000F1 (-S0e------------------mq1111--sT)
0x8C02E874   >480E         ldc r8,sr
RomBug: od
MPU type = 4, Input radix = 16
Ram (hard) breakpoints
Show control regs OFF, Show FP regs ON in hex
Show MMU regs OFF
RomBug:
```

Instruction Disassembly

```
RomBug: di Main
main                  >012A           sts pr,r1
main+0x2              >63F3           mov r15,r3
main+0x4              >9001           mov.w @(main+0xA,pc),r0 (#0x438)
main+0x6              >A0013308       bra main+0xC
                                      sub r0,r3
main+0xA              >0438           ldtlb
main+0xC             >9001          mov.w @(main+0x12,pc),r0 (#0xffff800c)
main+0xE              >A0010009       bra main+0x14
                                      nop
main+0x12             >800C           mov.b r0,@(0xC,r0)
main+0x14             >02EE           mov.l @(r0,r14),r2
main+0x16             >3236           cmp.hi r3,r2
main+0x18             >8B05           bf main+0x26
main+0x1A             >9001           mov.w @(main+0x20,pc),r0 (#0x470)
main+0x1C             >A0010009       bra main+0x22
                                      nop
main+0x20             >0470           dc.sw 0x0470
main+0x22             >00030009       bsrf r0
                                      nop
main+0x26             >62F3           mov r15,r2
```

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: 00000A04 8C16FEC8 8C001360 0000000C  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
_btext               C 8C048B70   btext                C 8C048B70
_bname               C 8C048BC8   bname                C 8C048BC8
main                 C 8C048BD0   init                 C 8C048BD8
config               C 8C048D8A   irqsort              C 8C048F10
input_irq            C 8C048FDC   output_irq           C 8C049286
status_irq           C 8C0493D0   entxirq              C 8C04944E
read                 C 8C0494E0   getstat              C 8C0494E8
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 8C0497AA
_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    0x8C048BD8    0      Word   ----------   ----------
01     ------------------------  Not  Set  ------------------------
RomBug: g
***Warning*** - breakpoints halt timesharing.
[33]$ iniz t3
<at watchpoint>
r0: 00000234 8C048BD8 000000C4 8FFEC2A4  8FFC8AE0 8FFC8AE0 8C0419DA 8FFEC2A0
r8: 00000000 8FFBED70 8FFC8AE0 8FFEC2A0  8FFC8890 00000002 8FFC8890 8FFC18B4
pr: 8C041A50 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C048BD8 sr: 40008000 (-S0e------------------mq0000--sf)
init                >2F96            mov.l r9,@-r15
RomBug: ow

WP #   ACCESS   ADDRESS      XBITS   SIZE   DATA         MASK
===================================================================
00     Instr    0x8C048BD8    0      Word   ----------   ----------
01     ------------------------  Not  Set  ------------------------
RomBug: g
```

```
<at watchpoint>
r0: 00000234 8C048BD8 000000C4 8FFEC2A4  8FFC8AE0 8FFC8AE0 8C0419DA 8FFEC2A0
r8: 00000000 8FFBED70 8FFC8AE0 8FFEC2A0  8FFC8890 00000002 8FFC8890 8FFC18B4
pr: 8C041A50 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C048BD8 sr: 40008000 (-S0e------------------mq0000--sf)
init                 >2F96             mov.l r9,@-r15
RomBug: ok
RomBug: ow

WP #   ACCESS   ADDRESS      XBITS   SIZE   DATA         MASK
===================================================================
00     ------------------------  Not  Set  ------------------------
01     ------------------------  Not  Set  ------------------------
RomBug: g
[34]$
[34]$
[34]$ break
<Called>
r0: 00000A04 8C16FEC8 8C001360 0000000C  00000000 00000000 8FFBED2C 00000000
r8: 40008001 00000000 00000000 00000000  00000000 00000000 8C008A40 8FFBECF0
pr: 8C02E6C4 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C048BD8 sr: 40008000 (-S0e------------------mq0000--sf)
init                 >2F96             mov.l r9,@-r15
RomBug: ok
RomBug: ow

WP #   ACCESS   ADDRESS      XBITS   SIZE   DATA         MASK
===================================================================
00     ------------------------  Not  Set  ------------------------
01     ------------------------  Not  Set  ------------------------
RomBug: g
[34]$
[34]$
```

Watch Point Example 2

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

```
[64]$ break
<Called>
r0: 00000A04 8C16FEC8 8C001360 0000000C  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: sd
_m_share             D 8FFC8890   drvrstatic           D 8FFC8890
_bdata               D 8FFC8890   bdata                D 8FFC8890
stopbits             D 8FFC8908   lu_list              D 8FFC890C
paritys              D 8FFC8910   wordsizes            D 8FFC8915
baudrates            D 8FFC891E   _enddata             D 8FFC8950
```

```
end                 D 8FFC8950   _jmptbl            D 8FFC8950
RomBug: r *
RomBug: g
***Warning*** - breakpoints halt timesharing.
[65]$
[65]$
[65]$ break
<Called>
r0: 00000A04 8C16FEC8 8C001360 0000000C  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 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 (#0xffffff68)
main+0x2            >012A             sts pr,r1
main+0x4            >33FC             add r15,r3
main+0x6            >909C             mov.w @(main+0x142,pc),r0 (#0xffff8050)
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
                                      nop
main+0x14           >2F96             mov.l r9,@-r15
main+0x16           >2F86             mov.l r8,@-r15
main+0x18           >6F33             mov r3,r15
main+0x1A           >2F12             mov.l r1,@r15
main+0x1C           >E900             mov #0x0,r9
main+0x1E           >E648             mov #0x48,r6
main+0x20           >7648             add #0x48,r6
dis: owi ffb5d9a
Watch Point # 0 set.
RomBug: ow

WP #   ACCESS   ADDRESS      XBITS   SIZE   DATA         MASK
===================================================================
00     Instr    0x0FFB5D9A    0      Word   ----------   ----------
01     ------------------------   Not  Set  ------------------------
RomBug: g
***Warning*** - breakpoints halt timesharing.
[66]$
[66]$ f_strap
<at watchpoint>
r0: 0000048A 0C16C908 00002000 0C16EB48  00000001 0C16EB48 0C16EB50 0C16C908
r8: 0C16EB90 00000000 00000000 00000000  00000000 00000000 0C173FFC 0C16EB04
pr: 0FFB5910 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 0FFB5D9A sr: 00008001 (-u0e------------------mq0000--sT)
main                >93AE             mov.w @(main+0x160,pc),r3 (#0xffffff68)
RomBug: g
```

```
"f_strap"
SUCCESS

[67]$

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

[72]$ break
<Called>
r0: 00000A04 8C16FEC8 8C001360 0000000C  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 kernel
RomBug: d .r14+80 10
0x8C008AC0           - 8FFBF4A0 8C00A470 00000000 8C008A74  .{t ..$p.......t
dis: oww.l .r14+80
Watch Point # 0 set.
RomBug: ow

WP #   ACCESS   ADDRESS      XBITS   SIZE   DATA         MASK
====================================================================
00     Write    0x8C008AC0    0      Long   ----------   ----------
01     ------------------------  Not  Set  ------------------------
RomBug: g
***Warning*** - br<at watchpoint>
r0: 00000084 8C008AC0 8C00A470 8FFC1CC8  00000004 8C00C7B0 00000002 0000000C
r8: 00000002 8FFED1F0 00000000 00000000  00000000 00000000 8C008A40 8C00C7A8
pr: 8C02B294 mach: 00000000 macl: 00000000 gbr: 00000000
pc: 8C02B2BA sr: 40008001 (-S0e------------------mq0000--sT)
rtnprc+0x3C         >E100             mov #0x0,r1
RomBug: d .r14+80 10
_jmptbl+0x80         - 8C00A470 8C00A470 00000000 8C008A74  ..$p..$p.......t
dis: ok
RomBug: ow

WP #   ACCESS   ADDRESS      XBITS   SIZE   DATA         MASK
====================================================================
00     ------------------------  Not  Set  ------------------------
01     ------------------------  Not  Set  ------------------------
RomBug: g
eakpoints halt timesharing.
[73]$
```

Chapter 8: 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 8-1 79R4700 RomBug Options

Option	Description
`b<n>`	Numeric input base radix
`r`	Use ROM type (soft) breakpoints (not supported by the 79R4700)
`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
`x`	Toggle disassembly hex output format

Table 8-2 79R4700 MIPS Options

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

Commands

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

Table 8-3 79R4700 Commands

Command	Description
e	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

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 8-4 79R4700 General Purpose Registers

Register Name	RomBug Name	Description
`r0`	`zero`	Constant Zero.
`r1`	`at`	Assembler temporary storage.
`r2-r3`	`v0-v1`	Function return.
`r4-r7`	`a0-a3`	Incoming args.
`r8-r15`	`t0-t7`	Registers for temporaries
`r16-r23`	`s0-s7`	Saved temporaries
`r24-r25`	`t8-t9`	Registers for temporaries
`r26-27`	`k0-k1`	Exception handling
`r28`	`gp`	Global data pointer
`r29`	`sp`	Stack pointer

Table 8-4 79R4700 General Purpose Registers (continued)

Register Name	RomBug Name	Description
`r30`	`cp`	Saved temporary
`r31`	`ra`	Return Address

Multiply and Divide Registers

Table 8-5 79R4700 Multiply and Divide Registers

Register Name	RomBug Name	Description
`HI`	`hi`	Multiply and Divide register Higher result
`LO`	`lo`	Multiply and Divide register Lower result

Program Counter Register

Table 8-6 79R4700 Program Counter Register

Register Name	RomBug Name	Description
`PC`	`pc`	Program Counter register

System Control Registers

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

Table 8-7 79R4700 System Control Registers

Register Number	Register Name	Description
0	`Index`	Programmable Pointer into TLB array
1	`Random`	Pseudorandom Pointer into TLB array (read only)
2	`EntryLo0`	Low half of TLB entry for even virtual page (VPN)
3	`EntryLo1`	Low half of TLB entry for odd virtual page (VPN)
4	`context`	Pointer to kernel virtual page table entry (PTE) for 32-bit address spaces
5	`PageMask`	TLB Page Mask
6	`Wired`	Number of wired TLB entries
7	`---`	Reserved
8	`BadVaddr`	Bad virtual address
9	`Count`	Timer Count
10	`EntryHi`	High half of TLB entry

Table 8-7 79R4700 System Control Registers

Register Number	Register Name	Description
11	`Compare`	Timer Compare
12	`SR`	Status Register
13	`Cause`	Cause of last exception
14	`EPC`	Exception Program Counter
15	`PRId`	Processor Revision Identifier
16	`Config`	Configuration register
17	`LLAddr`	Load Linked Address
18-19	`---`	Reserved
20	`XContext`	Pointer to kernel virtual PTE table for 64-bit address spaces
21-25	`---`	Reserved
26	`ECC`	Secondary-cache error checking and correcting (ECC) and Primary parity
27	`CacheErr`	Cache Error and Status register
28	`TagLo`	Cache Tag register
29	`TagHi`	Cache Tag register
30	`ErrorEPC`	Error Exception Program Counter
31	`---`	Reserved

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 8-8 79R4700 Floating Point Registers FR = 0

Floating-Point Register (FPR) Names
`df0`
`df1`
`df2`
`df3`
¦
`df28`
`df29`
`df30`
`df31`

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
```

Status Register Display

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

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

Table 8-9 Status Register Bit Field Assignments

Bit Field Number	Bit Field Name	Description
0	IE	Interrupt Enable. D = disabled, E = enabled.
1	EXL	Exception Level. N= normal, K = exception.
2	ERL	Error Level. N = normal, E = error.

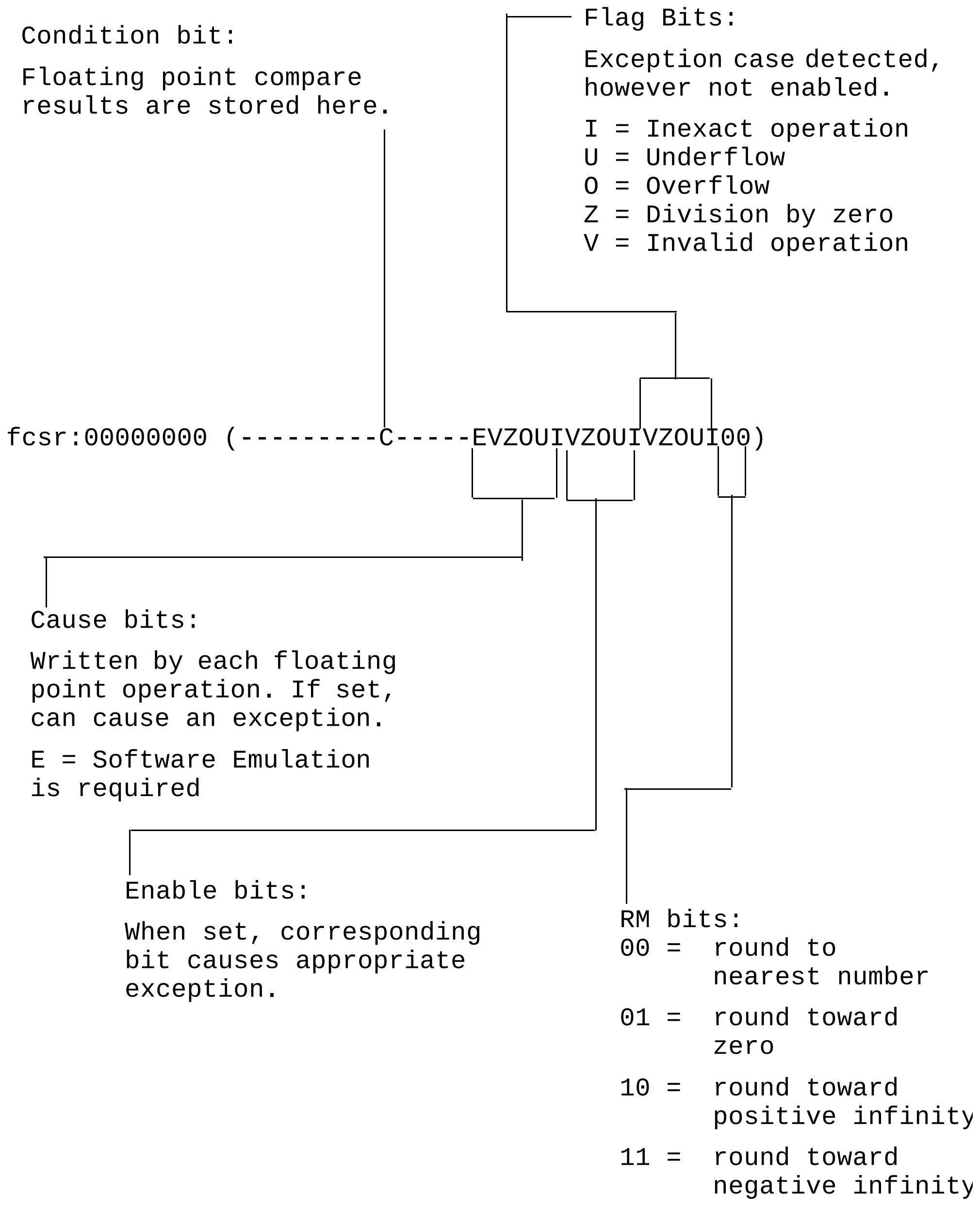
Table 8-9 Status Register Bit Field Assignments

Bit Field Number	Bit Field Name	Description
3-4	KSU	Mode bits. UR = User, SU = Supervisor, KK = Kernel
5	UX	Enables 64-bit virtual addressing and operations in User mode. 0 = 32-bit, x = 64-bit
6	SX	Enables 64-bit virtual addressing and operations in Supervisor mode. 0 = 32-bit, x = 64-bit
7	KX	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
8-15	IM	Interrupt Mask. Controls disabling and enabling interrupts.
16	DE	Specifies that cache parity errors cannot cause exceptions. E = enabled, D = Disabled.
17	CE	Must be 0.
18	CH	Read only.
19	0	Reserved. Always reads zeros. Must have zeros written to it.
20	SR	Read only.
21	0	Reserved. Always reads zeros. Must have zeros written to it.

Table 8-9 Status Register Bit Field Assignments

Bit Field Number	Bit Field Name	Description
22	BEV	Do not set. 0 indicates normal placement of vectors code
23-24	0	Reserved. Always reads zeros. Must have zeros written to it.
25	RE	Do not set. 0 indicates normal operation.
26	FR	Enable more floating point registers.
27	0	Reserved. Always reads zeros. Must have zeros written to it.
28-31	CU	Controls the usability of each of the four coprocessor unit numbers. U = usable

Floating Point Status and Control Register (FCR31)



```
fcsr:00000000 (---------C-----EVZOUIVZOUIVZOUI00)
```

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

Cause bits:

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

E = Software Emulation is required

Enable bits:

When set, corresponding bit causes appropriate exception.

RM bits:

00 =	round to nearest number
01 =	round toward zero
10 =	round toward positive infinity
11 =	round toward negative infinity

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:FFFFFFFF803caf20 a3:0000000000000020 t0:FFFFFFFF801a1900
t1:FFFFFFFFfffffffe t2:0000000014017b00 t3:0000000000000001 t4:0000000000000000
t5:0000000000000001 t6:00000000001c6ff0 t7:00000000001c6ff0 s0:0000000014017b01
s1:0000000000000000 s2:0000000000000000 s3:0000000000000000 s4:0000000000000000
s5:0000000000000000 s6:0000000000000000 s7:0000000000000000 t8:FFFFFFFF803c6b0c
t9:FFFFFFFF801b8378 k0:00000000000be060 k1:FFFFFFFF803caea8 gp:FFFFFFFF801b0000
sp:FFFFFFFF803caea8 cp:0000000000000000 ra:FFFFFFFF80070eb0 hi:001C2F70001c2ec8
lo:0000000000000000 pc:FFFFFFFF80070eb0
sr:14017B00 (---U-F---------DNIIIINSSxxxKKNND)
0x80070EB0   >40906000         mtc0       s0,$12
RomBug: a spsonic
RomBug: b dbg6
RomBug: b dbgR
RomBug: g
***Warning*** - breakpoints halt timesharing.
[2]$
[2]$
[2]$ ipstart
<at breakpoint>
at:FFFFFFFF8007cd58 v0:0000000000000080 v1:FFFFFFFF801171b0 a0:FFFFFFFFbf60001c
a1:00000000000001c4 a2:FFFFFFFFffffffff a3:0000000000008000 t0:0000000000000001
t1:0000000000000000 t2:0000000000000003 t3:0000000000000028 t4:0000000014017b01
t5:FFFFFFFF803fd684 t6:0000000000000000 t7:0000000000000000 s0:FFFFFFFF801cc500
s1:FFFFFFFF803ee350 s2:FFFFFFFF801cc660 s3:FFFFFFFF801cc570 s4:FFFFFFFF80144520
s5:FFFFFFFF801b0000 s6:FFFFFFFF801cc500 s7:FFFFFFFF803ca87c t8:FFFFFFFF80144ede
t9:FFFFFFFF801b79a8 k0:0000000014017b01 k1:FFFFFFFFffffffc1 gp:FFFFFFFF803ee350
sp:FFFFFFFF803ca670 cp:FFFFFFFF8013f1d0 ra:FFFFFFFF80141e6c hi:0000000000000000
lo:0000000000000078 pc:FFFFFFFF80141e80
sr:14017B03 (---U-F---------DNIIIINSSxxxKKNKE)
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:FFFFFFFF801171b0 a0:FFFFFFFF801cc5b8
a1:00000000000001c4 a2:FFFFFFFFffffffff a3:0000000000008000 t0:0000000000000001
t1:0000000000000000 t2:0000000000000003 t3:0000000000000028 t4:0000000014017b01
t5:FFFFFFFF803fd684 t6:0000000000000000 t7:0000000000000000 s0:FFFFFFFF801cc500
s1:FFFFFFFF803ee350 s2:FFFFFFFF801cc660 s3:FFFFFFFF801cc570 s4:FFFFFFFF80144520
s5:FFFFFFFF801b0000 s6:FFFFFFFF801cc500 s7:FFFFFFFF803ca87c t8:FFFFFFFF80144ede
t9:FFFFFFFF801b79a8 k0:0000000014017b01 k1:FFFFFFFFffffffc1 gp:FFFFFFFF803ee350
sp:FFFFFFFF803ca670 cp:FFFFFFFF8013f1d0 ra:FFFFFFFF80141e6c hi:0000000000000000
lo:0000000000000078 pc:FFFFFFFF80141e84
sr:14017B03 (---U-F---------DNIIIINSSxxxKKNKE)
dbg6+0x4            >8FA50008         lw        a1,0x8(sp)
trace: d .a0
0x801CC5B8           - BF600000 A0370000 A0370040 00000004  ?‘.. 7.. 7.@....
0x801CC5C8           - A03700C0 00000001 A03701C4 00000004   7.@.... 7.D....
0x801CC5D8           - 02000000 002A0000 00000000 00000000  .....*..........
0x801CC5E8           - 00000000 00000000 00000000 00000000  ................
0x801CC5F8           - 00000000 80370000 00010000 00000000  .....7..........
0x801CC608           - 00000000 00000000 00000000 02000000  ................
0x801CC618           - 002A0000 00008000 00008000 00000043  .*.............C
0x801CC628           - 0000000F 00000004 00000004 80381004  .............8..
0x801CC638           - 6462675F 73707369 6E000000 00000000  dbg_spsin.......
0x801CC648           - 801B0000 00000000 00000000 00000000  ................
0x801CC658           - 00000000 00000000 000000FF 80102B1C  ..............+.
0x801CC668           - 8013F680 8013F90C 8013FD44 8013FE44  ..v...y...}D..~D
0x801CC678           - 8013FA20 8013FA00 00000001 00000000  ..z ..z.........
0x801CC688           - 00000000 00000000 00000000 00000000  ................
0x801CC698           - 80086944 00000000 00000000 00000000  ..iD............
0x801CC6A8           - 00000000 00000000 00000000 00000000  ................
dis: .rr1 .a0
RomBug: d .rr1
0x80000000+rr1       - BF600000 A0370000 A0370040 00000004  ?‘.. 7.. 7.@....
0x80000010+rr1       - A03700C0 00000001 A03701C4 00000004   7.@.... 7.D....
0x80000020+rr1       - 02000000 002A0000 00000000 00000000  .....*..........
0x80000030+rr1       - 00000000 00000000 00000000 00000000  ................
0x80000040+rr1       - 00000000 80370000 00010000 00000000  .....7..........
0x80000050+rr1       - 00000000 00000000 00000000 02000000  ................
0x80000060+rr1       - 002A0000 00008000 00008000 00000043  .*.............C
0x80000070+rr1       - 0000000F 00000004 00000004 80381004  .............8..
0x80000080+rr1       - 6462675F 73707369 6E000000 00000000  dbg_spsin.......
0x80000090+rr1       - 801B0000 00000000 00000000 00000000  ................
0x800000A0+rr1       - 00000000 00000000 000000FF 80102B1C  ..............+.
0x800000B0+rr1       - 8013F680 8013F90C 8013FD44 8013FE44  ..v...y...}D..~D
0x800000C0+rr1       - 8013FA20 8013FA00 00000001 00000000  ..z ..z.........
0x800000D0+rr1       - 00000000 00000000 00000000 00000000  ................
0x800000E0+rr1       - 80086944 00000000 00000000 00000000  ..iD............
0x800000F0+rr1       - 00000000 00000000 00000000 00000000  ................
```

```
dis: .rr3 bf600000
RomBug: .rr2 [.rr1+4]
RomBug: d .rr2
0x80000000+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000010+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000020+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000030+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000040+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000050+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000060+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000070+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000080+rr2       - 00000000 00000000 00000000 00000000  ................
0x80000090+rr2       - 00000000 00000000 00000000 00000000  ................
0x800000A0+rr2       - 00000000 00000000 00000000 00000000  ................
0x800000B0+rr2       - 00000000 00000000 00000000 00000000  ................
0x800000C0+rr2       - 00000000 00000000 00000000 00000000  ................
0x800000D0+rr2       - 00000000 00000000 00000000 00000000  ................
0x800000E0+rr2       - 00000000 00000000 00000000 00000000  ................
0x800000F0+rr2       - 00000000 00000000 00000000 00000000  ................
```

Chapter 9: SH-5 Processors

The following is provided in this section for SH-5 family processors:

- **-o 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

Option	Description
`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 <n>. If no vector specified, displays current set of monitored vectors. `'-'` = disable monitoring `'s'` = system state only `'u'` = user state only `'d'` = display only `<m>` = end of range of vectors
`v?`	Display all exception vector values

Table 9-2 SH-5 Options

Option	Description
`.`	Display registers upon monitored exception
`a`	Toggle control registers display
`c [<num> <value>]`	Set a watchpoint trigger event counter. If no `<num>` nor `<value>` is specified, all the watchpoint trigger event counters will be printed. The SH8001 has one event counter numbered zero.
`d`	Toggle floating-point register display (decimal format)
`f`	Toggle floating-point register display (hex format)
`k [<watch_num>]`	Kill watchpoint(s) `<watch_num>`: watchpoint number If `<watch_num>` not present, kill all watchpoints
`m`	Toggle MMU registers display
`tw`	Trace over an operand access or instruction value watchpoint trigger.
`w[<ia|oa|iv>]`	Display table of active watchpoints, ia specifes instruction access, oa specifies operand access, and iv specifies instruction value.

Table 9-2 SH-5 Options

Option	Description
`w[ia|oa] <start> <end> [a<n>] [i<m|c|b>] [s<s|u|b>] [c<n>]`	Set an instruction access (ia) or operand access (oa) watchpoint range. `a<n>` = ASID value `i<m|c|b>` = ISA (m)edia, (c)ompact, or (b)oth `s<s|u|b>` = state (s)ystem, (u)ser, or (b)oth `c<n>` = event counter <n> SH8001 has four available instruction access ranges and two available operand ranges. It also has one event counter numbered zero (see c).
`wiv <value> <mask> [a<n>] [s<s|u|b>] [c<n>]`	Set an instruction value watchpoint. `<value>` is the value to watch for. `<mask>` is the mask to apply to instructions before checking against `<value>`. `a<n>` = ASID value `s<s|u|b>` = state (s)ystem, (u)ser, or (b)oth `c<n>` = event counter <n> SH8001 has two available instruction value watchpoints. It also has one event counter numbered zero (see c).

Commands

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

Table 9-3 79R4700 Commands

Command	Description
e	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

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

Register Name	RomBug Name	Description
`r0,r1`		Caller-save registers
`r2-r9`		Incoming parameter registers, r2 is return value
`r10`	`lr`	Link register
`r11`	`at`	Assembler/linker temporary register
`r12`	`fp`	Frame pointer
`r13`	`cp`	Constant data area pointer
`r14`	`gp`	Global data area pointer
`r15`	`sp`	Stack pointer
`r16-r19`		Caller-save registers
`r20-r23`		Callee-save registers

Table 9-4 General Purpose Registers

Register Name	RomBug Name	Description
`r24-r31`		Caller-save registers
`r32-r62`		Callee-save registers
`r63`	`zero`	Hard-coded zero value register

Program Counter Register

Table 9-5 SH-5 Program Counter Register

Register Name	RomBug Name	Description
`PC`	`pc`	Program counter register

System Control Registers

There are 64 64-bit control registers.

Table 9-6 SH-5 System Control Registers

Register Number	Register Name	Description
0	`sr`	Status register
1	`ssr`	Saved status register
2	`pssr`	Panic-saved status register
4	`intenv`	Interrupt event ID
5	`expevt`	Exception event ID
6	`pexpevt`	Panic-saved exception event ID
7	`tra`	TRAP exception number
8	`spc`	Saved program counter
9	`pspc`	Panic-saved saved program counter
10	`resvec`	Reset vector
11	`vbr`	Vector base register
13	`tea`	Faulting effective address
16	`dcr`	Debug control register
17	`kcr0`	Kernel control register 0

Table 9-6 SH-5 System Control Registers

Register Number	Register Name	Description
18	`kcr1`	Kernel control register 1
62	`ctc`	Clock tick counter
63	`usr`	User status register

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 9-7 SH-5 Floating-point registers

Register Names	Description
`fr0 - fr11`	Incoming floating-point parameters, fr0 and fr1 are floating-point return registers
`fr12 - fr15`	Callee-save registers
`fr16 - fr35`	Caller-save registers
`fr36 - fr63`	Callee-save registers

Target Address Registers

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

Table 9-8 SH-5 Target address registers

Register Names	Description
`tr0 - tr4`	Caller-save target address registers
`tr5 - tr8`	Callee-save target address registers

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
```

Status Register Display

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

```
sr: 400080F0 (-S------++asid++D-----mqimsk--s-)
                bit 31                            bit 0
```

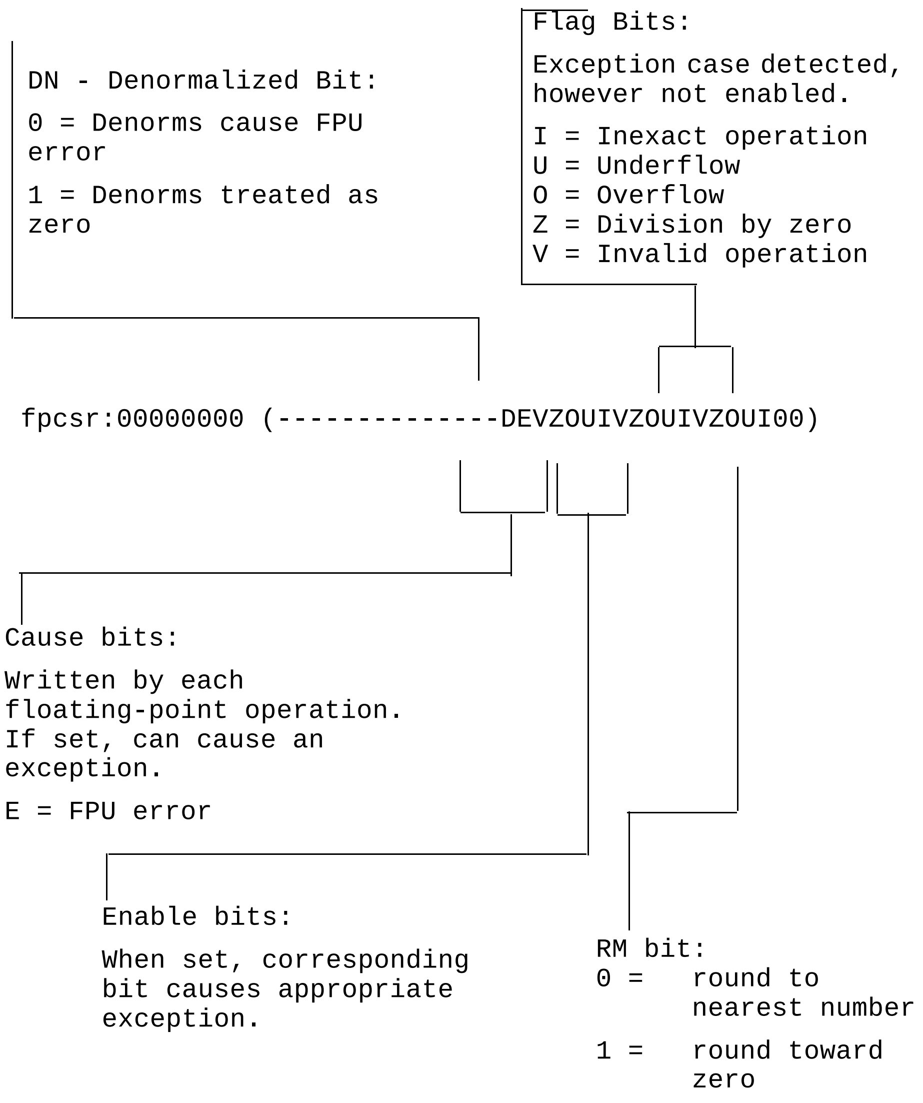
Table 9-9 Status Register Bit Field Assignments

Bit Field Number	Bit Field Name	Description
1	S	Saturation control for SHcompact integer instructions (S = set, s = clear)
4-7	IMASK	Interrupt mask level (always displays imsk)
8	Q	Divide step state for SHcompact integer instructions (Q = set, q = clear)

Table 9-9 Status Register Bit Field Assignments

Bit Field Number	Bit Field Name	Description
9	M	Divide step state for SHcompact integer instructions (M = set, m = clear)
11	CD	Clock tick counter disable (C = set, - = clear)
12	PR	Precision for SHcompact floating-point instructions (P = set, - = clear)
13	SZ	Size for SHcompact floating-point instructions (S = set, - = clear)
14	FR	Floating-point register bank for SHcompact floating-point instructions (R = set, - = clear)
15	FD	Floating-point disable (D = set, - = clear)
16-23	ASID	Address space ID (Always displays ++asid++)
26	WATCH	Watchpoint enable flag (W = set, - = clear)
27	STEP	Single-step enable flag (S = set, - = clear)
28	BL	Event block flag (B = set, - = clear)
30	MD	Mode bit (S = set, system-state, U = clear, user-state)
31	MMU	MMU enable flag (M = set, - = clear)

Floating-point Status and Control Register (FCR31)



DN - Denormalized Bit:

0 = Denorms cause FPU error

1 = Denorms treated as zero

Flag Bits:

Exception case detected, however not enabled.

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

```
fpcsr:00000000 (--------------DEVZOUIVZOUIVZOUI00)
```

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 zero

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>
r00: 0000000040041d71 0000000040041d71  0000000040041498 0000000040041000
r04: 0000000040041c50 0000000040041498  0000000000000000 0000000040041810
r08: 0000000000000000 0000000000000000  0000000000001295 0000000000000015
r12: 0000000000000000 0000000000007ff0  0000000040048ff0 0000000040064f98
r16: 0000000000000000 0000000000000000  0000000000000000 0000000000000000
r20: 0000000040041810 00000000400418e0  000000004004185c 0000000000000001
r24: 0000000000000000 0000000000000000  0000000000000000 0000000000000001
r28: 000000000000003f 000000000000003e  0000000000000008 0000000000000000
r32: 0000000040041000 0000000000000000  0000000000024000 0000000040041000
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: 40041D71 000081FD 000081DD 00000000  00000000 00000000 00000000 00000000
sr: 400080F0 (-S------++asid++D-----mqimsk--s-)
pc: 00001295
0x00001294   >89402800         ld.l r20,40,r0
RomBug: b 12a0
RomBug: g
<at breakpoint>
r00: 00000000400494d4 0000000040041d71  0000000040041498 0000000040041000
r04: 0000000040041c50 0000000040041498  0000000000000000 0000000040041810
r08: 0000000000000000 0000000000000000  0000000000001295 000000000000005d
r12: 0000000000000000 0000000000007ff0  0000000040048ff0 0000000040064f98
r16: 0000000000000000 0000000000000000  0000000000000000 0000000000000000
r20: 0000000040041810 00000000400418e0  000000004004185c 0000000000000001
r24: 0000000000000000 0000000000000000  0000000000000000 0000000000000001
r28: 000000000000003f 000000000000003e  0000000000000008 0000000000000000
r32: 0000000040041000 0000000000000000  0000000000024000 0000000040041000
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: 40041D71 000081FD 000081DD 00000000  00000000 00000000 00000000 00000000
sr: 400080F0 (-S------++asid++D-----mqimsk--s-)
pc: 000012A1
0x000012A0   >6BF52E00         ptrel r11,tr0 (0x12FE)
RomBug:
```

Trace Command

The following example illustrates the trace and memory display commands.

```
r00: 00000000400494d4 0000000040041d71  0000000040041498 0000000040041000
r04: 0000000040041c50 0000000040041498  0000000000000000 0000000040041810
r08: 0000000000000000 0000000000000000  0000000000001295 000000000000005d
r12: 0000000000000000 0000000000007ff0  0000000040048ff0 0000000040064f98
r16: 0000000000000000 0000000000000000  0000000000000000 0000000000000000
r20: 0000000040041810 00000000400418e0  000000004004185c 0000000000000001
r24: 0000000000000000 0000000000000000  0000000000000000 0000000000000001
r28: 000000000000003f 000000000000003e  0000000000000008 0000000000000000
r32: 0000000040041000 0000000000000000  0000000000024000 0000000040041000
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: 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............
0x40049514    - 00000000 00000000 00000000 00000000  ................
dis: .rr1 .r0
RomBug: t
r00: 00000000400494d4 000000000002b535  0000000040041498 0000000040041000
r04: 0000000040041c50 0000000040041498  0000000000000000 0000000040041810
r08: 0000000000000000 0000000000000000  0000000000001295 000000000000005d
r12: 0000000000000000 0000000000007ff0  0000000040048ff0 0000000040064f98
r16: 0000000000000000 0000000000000000  0000000000000000 0000000000000000
r20: 0000000040041810 00000000400418e0  000000004004185c 0000000000000001
r24: 0000000000000000 0000000000000000  0000000000000000 0000000000000001
r28: 000000000000003f 000000000000003e  0000000000000008 0000000000000000
r32: 0000000040041000 0000000000000000  0000000000024000 0000000040041000
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 .rr1
0x00000000+rr1 - A8900001 00000000 400498E0 0002B535  (.......@..`..55
0x00000010+rr1 - 0002C299 00000000 00000000 00000000  ..B.............
0x00000020+rr1 - 00000000 00000000 00000000 0002F461  ..............ta
0x00000030+rr1 - 400494D0 00000000 00000000 00000000  @..P............
0x00000040+rr1 - 00000000 00000000 00000000 00000000  ................
dis:
```

Index

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:____

Company:____

Phone:____

Fax:_______________________Email:_____________________

Product Name: ____

Description of Problem:

____

____

____

____

____

____

____

____

____

____

Host Platform____

Target Platform_______________________________________