User's Manual

RENESAS

SuperH™ RISC engine Simulator/Debugger

User's Manual

Renesas Microcomputer Development Environment System

SPARC: Solaris,
HP9000 Series 700

Renesas Electronics
www.renesas.com

Rev.1.0 1999.09

Cautions

Preface

The SuperH™ RISC engine Simulator/Debugger (referred to in this manual as the simulator/debugger) is a software tool that simulates the SuperH™ RISC engine series of microcomputers to support software development on a host computer.

This manual describes the simulator/debugger overview and its usage. Carefully read this manual before using the simulator/debugger. For the C/C++ compiler, assembler, inter-module optimizer, and librarian related to this simulator/debugger, read the following manuals.

- SuperH™ RISC engine C/C++ Compiler User's Manual
- SuperH™ RISC engine Assembler User's Manual
- Linkage Editor, Librarian, Object Converter User's Manual

For details on each SuperH™ RISC engine series microprocessors, refer to appropriate hardware and programming manuals.

This manual explains the overview of the debugger and how to set up and operate it. For detailed operation, initiate the simulator/debugger to read the on-line manual.

It is assumed that commands are input from the workstation after the C shell is initiated. When using another shell, refer to the host system and other related manuals.

The following symbols are used in this manual:

< >:	The contents within < > are to be specified.
[ ]:	Parameters enclosed with [ ] can be omitted.
{A|B}:	Either A or B can be selected.
Δ:	Indicates one or more blank spaces.
(SP):	Press the space key.
(RET):	Press the return key.
Input:	The bold italic face indicates the input by the user.
%:	C shell prompt.

Contents

Figures

Tables

Section 1 Overview

This simulator/debugger provides the CPU simulation and debugging function for SuperH™ RISC engine series microcomputers to support simulation for the SH-1, SH-2, SH-3, SH-4, SH-2E, SH-3E and SH-DSP series. The simulator/debugger Ver. 2 promotes efficient debugging of programs written in the C++ language in addition to those written in the C or assembly language.

This simulator/debugger operates together with the user interface software that runs on the workstation.

1.1 Operating Environment

This simulator/debugger supports the following machine environments as a host system.

(1) Machine with SPARC*[1] (hereinafter referred to as SPARC)

Operating system (OS):	Solaris version*[2] 2.4 (OSF/Motif*[3])
Window system:	OSF/Motif
Memory capacity:	32 Mbytes or more (differs depending on the system operating status)
Disk capacity:	30 Mbytes or more (including free space required for performing operations)

(2) HP9000 series 700*[4] (hereinafter referred to as HP9000)

Operating system (OS):	HP-UX 10.2*[5]
Window system:	OSF/Motif
Memory capacity:	32 Mbytes or more (differs depending on the system operating status)
Disk capacity:	30 Mbytes or more (including free space required for performing operations)

(3) Software configuration

The simulator/debugger is configured as follows:

Installer: cas_install

Interface software: csdsh, dbgif

Simulator/debugger: sdsh12, sdshdsp, sdsh3e, sdsh2e, sdsh4

CPU information file creation program: ciash, ciashdsp, ciash4

Notes: 1. SPARC is a CPU and workstation administrated by SPARC International, Inc., and is based on the architecture developed by Sun Microsystems, Inc. (United States).

2. Solaris is a trademark of Sun Microsystems, Inc. (United States).
3. OSF/Motif is a trademark of Open Software Foundation, Inc. (United States).
4. HP9000 Series 700 is a trademark of Hewlett-Packard Company. (United States).
5. HP-UX is a trademark of Hewlett-Packard Company. (United States).

1.2 Features

(1) Since the simulator/debugger runs on a host computer, software debugging can start without using an actual SuperH™ RISC engine user system, thus reducing overall system development time.

(2) The simulator/debugger performs a pipeline simulation to calculate the number of instruction execution cycles for a program, thus enabling performance evaluation without using an actual SuperH™ RISC engine user system.

(3) The simulator/debugger offers the following features and functions that enable efficient program testing and debugging.

- The ability to handle SuperH™ RISC engine CPUs
- Functions to trace instructions or subroutines
- Functions to stop or continue execution when an error occurs during user program execution
- Function-unit performance measurement
- A comprehensive set of break functions
- Functions to define and modify memory areas

1.3 Simulation Range

(1) The simulator/debugger supports the following SuperH™ RISC engine microcomputer functions.

- All execution instructions (pipeline simulation)
- Exception processing
- Registers
- All address areas
- MMU (only for SH-3, SH-3E, and SH-4 series)
- Cache (only for SH-3, SH-3E, and SH-4 series)
- DMAC (only for SH-4 series)
- BSC (only for SH-4 series)
- FPU (only for SH-2E, SH-3E, and SH-4 series)

(2) The simulator/debugger does not support the following SuperH™ RISC engine microcomputer functions. Programs that use these functions must be debugged with the SuperH™ RISC engine emulator.

- Timer
- Serial communication interface
- I/O port
- Interrupt controller (INTC)

1.4 Notes

(1) When loads to the host computer is large, such as when many processes are operating, the simulator/debugger cannot be initiated even when the memory capacity specified above is available.

(2) When using an X Window System*[6] terminal, it must be the one on which the window system for the host computer can operate.

(3) Terminating subwindow display

Subwindow display, such as Dump window display, cannot be terminated by clicking the Close or Cancel button. Click the STOP button of the base window and then the Close or Cancel button to close a subwindow.

(4) Terminating simulator/debugger

The simulator/debugger cannot be terminated during debugger command processing. Click the STOP button of the base window to abort the processing and then terminate the simulator/debugger.

(5) Re-initiating simulator/debugger

The simulator/debugger operates with two processes: csdsh and dbgif. Selecting Quit in the window menu forcibly terminates only the csdsh process, and or dbgif may remain. In this case, the integrated development manager cannot be re-initiated. Terminate dbgif in the following way:

Example: Check if dbgif process remains while the integrated development manager is not operating:

%***ps -e|grep dbgif(RET)***

  689 pts/4 1:32 dbgif

If dbgif process remains, forcibly terminate it with the kill command:

%***kill -9 <PID>(RET)***

<PID>: The process number displayed at the beginning of the ps command execution result (689 in the above example)

(6) If the following error message is output when the integrated development manager is initiated on SPARC, set with environment variable LD_LIBRARY_PATH the name of the directory in which the dynamic link library (libXt) is stored.

<Error message>

ld.so.x: csdsh: fatal: libXt.so.x: can't open file: error=2

— When environment variable LD_LIBRARY_PATH has not been specified

  %***setenv LD_LIBRARY_PATH <directory that stores library>(RET)***

— When environment variable LD_LIBRARY_PATH has been specified

  %***setenv LD_LIBRARY_PATH <directory specified before>:<directory that stores library>(RET)***

Notes: 6. X Window System is a product designed by Massachusetts Institute of Technology.

Section 2 Simulator/Debugger Functions

This section describes the SuperH™ RISC engine simulator/debugger Ver. 2. Note that the endian, MMU, cache, and control registers can be used only in the SH-3, SH-3E, and SH-4 series, and BSC and DMAC can be used only in the SH-4 series.

2.1 Simulator/Debugger Memory Management

(1) Memory Map Specification

A memory map is specified to calculate the number of memory access cycles during simulation. The simulator/debugger supports the memory types shown in table 2.1.

Table 2.1 Memory Types

Memory Type	User Program Execution
Internal ROM area (X-ROM, Y-ROM)	Enabled
Internal RAM area (X-RAM, Y-RAM)	Enabled
External bus area	Enabled
Internal I/O area	Disabled

A memory map is specified by a CPU information file. For how to create a CPU information file, refer to section 7, How to Create CPU Information File.

(3) Memory resource allocation

A memory resource is automatically allocated by loading the user program.

An area not defined in the program, however, is not allocated. In this case, allocate a memory resource using the MAP_SET command.

(4) SH-4 memory management

The bus width of area 0 and the size of MPX memory and NORMAL memory must be specified in the CPU information file. Other settings must be specified in the BSC register.

2.2 Endian

In the SH-3, SH-3E, and SH-4 series, little endian as well as big endian can be specified as the data allocation format in the memory; a user program created in the little endian format can also be simulated and debugged. The endian can be selected by the option when the simulator/debugger is started. For details on the option, refer to section 3.2, Start-up.

2.3 Pipeline Reset Processing

The simulator/debugger, which simulates the pipeline, resets the pipeline when:

- The program counter (PC) is modified after the instruction simulation stops and before it restarts.
- The GO command to which the execution start address has been specified is executed.
- Initialization is performed or a program is loaded.
- Memory data being currently fetched and decoded is rewritten.

When the pipeline is reset, data already fetched and decoded is cleared, and new data is fetched and decoded from the current PC. In addition, the number of executed instructions and the number of instruction execution cycles are cleared to zero.

2.4 Memory Management Unit (MMU)

For the SH3, SH3E, and SH-4 series, the simulator/debugger simulates MMU operations such as TLB operations, address translation, or MMU-related exceptions (TLB miss, TLB protection exception, TLB invalid exception, and initial page write). The user program using address translation by the MMU can be simulated and debugged. In addition, the MMU-related exception handler routines can be simulated and debugged.

As well as during user program execution, the MMU translates virtual addresses into physical addresses during address display or input in the dialog boxes or windows. Therefore, in the dialog boxes and windows, memory can be accessed with the virtual addresses used in the user program. The MMU operations depend on the MCU type.

2.5 Cache

For the SH-3, SH-3E, and SH-4 series, the simulator/debugger simulates operations of the cache. Cache operations during user program execution can be monitored.

In the simulator/debugger, the cache hit ratio can be displayed with the STATUS command.

Checking and Displaying the Cache Hit Ratio: The simulator/debugger displays the cache hit ratio in percentage with the STATUS command. The cache hit ratio is obtained by dividing the cache hit count by the cache access count (the sum of the cache hit count and cache miss count).

Initializing the Cache Hit Ratio: The displayed cache hit ratio is reset to zero when the simulator/debugger is initiated, the pipeline is reset, or the CCR register value is modified.

Note: The simulator/debugger does not change the high-order three bits of the address tag stored in a cache address array to zeros.
When loading memory to the area to which the cache has been mapped by selecting the [File Load] menu, turn the AT bit of the MMUCR off to disable the MMU.

2.6 Bus State Controller (BSC)

For the SH-4 series, the simulator/debugger has the functions for specifying and modifying the memory map to use the BSC; the user program using the BSC can be debugged.

Table 2.2 lists the memory types that can be specified for the SH-4 series.

Table 2.2 Memory Types for the SH-4 Series

Address	Specifiable Memory Types
H'00000000 to H'03FFFFFF (area 0)	Normal memory, burst ROM, and MPX
H'04000000 to H'07FFFFFF (area 1)	Normal memory, byte control SRAM, and MPX
H'08000000 to H'0BFFFFFF (area 2)	Normal memory, DRAM, SDRAM, and MPX
H'0C000000 to H'0FFFFFFF (area 3)	Normal memory, DRAM, SDRAM, and MPX
H'10000000 to H'13FFFFFF (area 4)	Normal memory, byte control SRAM, and MPX
H'14000000 to H'17FFFFFF (area 5)	Normal memory, burst ROM, and MPX
H'18000000 to H'1BFFFFFF (area 6)	Normal memory, burst ROM, and MPX
H'1C000000 to H'1FFFFFFF (area 7)	Cannot be specified
H'7C000000 to H'7C001FFF	Internal RAM (cannot be changed)
H'E0000000 to H'FFFFFFFF	I/O (cannot be changed)

The high-order three bits of the addresses for areas 0 to 7 in table 2.2 must be ignored; H'00000000 and H'20000000 are both in area 0.

The simulator/debugger does not support the PCMCIA.

2.7 Direct Memory Access Controller (DMAC)

For the SH-4 series, the simulator/debugger simulates the 4-channel DMAC operations; the user program using the DMAC can be debugged.

2.8 Exception Processing

The simulator/debugger detects the generation of exceptions corresponding to TRAPA instructions, general illegal instructions, slot illegal instructions, and address errors. In addition, for the SH-3, SH-3E, and SH-4 series, the simulator/debugger simulates MMU-related exception processing (TLB miss, TLB protection exception, TLB invalid exception, and initial page write). For the SH-2E, SH-3E, and SH-4 series, the simulator/debugger also simulates FPU exception processing. This also enables simulation when an exception occurs.

Exception processing is simulated as follows according to the execution mode selected by the EXEC_MODE command.

(1) SH-1, SH-2, SH-2E, and SH-DSP Series:

[1] When [C (continue)] is selected (continue mode):

(a) Detects an exception during instruction execution.

(b) Saves the PC and SR in the stack area.

(c) Reads the start address from the vector address corresponding to the vector number.

(d) Starts instruction execution from the start address. If the start address is 0, the simulator/debugger stops exception processing, displays that an exception processing error has occurred, and enters the command input wait state.

[2] When the [S (stop)] is selected (stop mode):

Executes steps (a) to (c) above, then stops.

(2) SH-3 and SH-3E Series:

[1] When [C (continue)] is selected (continue mode):

(a) Detects an exception during instruction execution.

(b) Saves the PC and SR to the SPC and SSR, respectively.

(c) Sets the BL bit, RB bit, and MD bit in the SR to 1s.

(d) Sets an exception code in control registers EXPEVT. If necessary, appropriate values are set in other control registers.

(e) Sets the PC to the vector address corresponding to the exception cause. (If an exception is detected when the BL bit in the SR is 1, reset vector address H'A0000000 is set regardless of the exception cause.)

(f) Starts instruction execution from the address set in the PC.

[2] When the [S (stop)] is selected (stop mode):

Executes steps (a) to (e) above, then stops.

(3) SH-4 Series:

[1] When [C (continue)] is selected (continue mode):

(a) Detects an exception during instruction execution.

(b) Saves the PC and SR to the SPC and SSR, respectively.

(c) Sets the BL bit, RB bit, and MD bit in the SR to 1s.

(d) Sets the FD (FPU disable) bit in the SR to 0 at reset.

(e) Sets an exception code in control register EXPEVT. If necessary, appropriate values are set in other control registers.

(f) Sets the PC to the vector address corresponding to the exception cause. (If an exception is detected when the BL bit in the SR is 1, reset vector address H'A0000000 is set regardless of the exception cause.)

(g) Starts instruction execution from the address set in the PC.

[2] When the [S (stop)] is selected (stop mode):

Executes steps (a) to (f) above, then stops.

2.9 Control Registers

For the SH-3, SH-3E, and SH-4 series, the simulator/debugger supports the memory-mapped control registers that are used for exception processing, MMU control, and cache control. In addition, for the SH-4 series, the simulator/debugger supports the control registers that are used for BSC control and DMAC control. Therefore, a user program using exception processing, MMU control, and cache control can be simulated and debugged.

MMU	PTEH:	Page table entry high register
	PTEL:	Page table entry low register
	TTB:	Translation table base register
	TEA:	TLB exception address register
	MMUCR:	MMU control register
Exception processing	TRA:	TRAPA exception register
	EXPEVT:	Exception event register
	INTEVT:	Interrupt event register
Cache	CCR:	Cache control register
	QACR0 and QACR1*:	Queue address control registers 0 and 1
BSC	BCR1 and BCR2*:	Bus control registers 1 and 2
	WCR1 to WCR3*:	Wait state control registers 1 to 3
	MCR*:	Individual memory control register
	RTCSR*:	Refresh timer control/status register
	RTCNT*:	Refresh timer/counter
	RTCOR*:	Refresh time constant register
	RFCR*:	Refresh count register
DMAC	SAR0 to SAR3*:	DMA source address registers 0 to 3
	DAR0 to DAR3*:	DMA destination address registers 0 to 3
	DMATCR0 to DMATCR3*:	DMA transfer count registers 0 to 3
	CHCR0 to CHCR3*:	DMA channel control registers 0 to 3
	DMAOR*:	DMA operation register

Note: The registers marked with * are supported only for the SH-4 series.

The simulator/debugger does not support the PCMCIA interface and the synchronous DRAM mode register.

2.10 Trace

The simulator/debugger writes the execution results into the trace buffer, which can hold the results for up to 1024 instructions. The trace information acquisition conditions are specified by the TRACE_CONDITION command. The acquired trace information is displayed on the Trace window.

The trace information displayed in the Trace window depends on the target CPU as follows.

(1) SH-1, SH-2, SH-2E, and SH-DSP Series:

- C/C++ or assembly-language source programs
- Total number of instruction execution cycles
- Instruction address
- Pipeline execution status
- Instruction mnemonic
- Data access information (destination and accessed data)

(2) SH-3 and SH-3E Series:

- C/C++ or assembly-language source programs
- Total number of instruction execution cycles
- Data on the address bus
- Data on the data bus
- Instruction code
- Instruction number
- Instruction mnemonic
- Instruction number that was fetched
- Instruction number that was decoded
- Instruction number that was executed
- Instruction number that accessed memory
- Instruction number that wrote back data
- Data access information (destination and accessed data)

(3) SH-4 Series:

- C/C++ or assembly-language source programs
- Total number of instruction execution cycles (CPU internal clock)
- Program counter address
- Instruction number that was fetched
- Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by EX pipeline simulation
- Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by LS pipeline simulation

— Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by BR pipeline simulation
— Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by FP pipeline simulation
— Instruction number assigned to the instruction to be executed
— Memory address, instruction code, and mnemonic of the instruction to be executed.
— Data access information (destination and accessed data)

2.11 Standard I/O and File I/O Processing

The simulator/debugger supports the standard I/O and file I/O processing to enable input and output between the user program and standard I/O devices (console and keyboard). The supported I/O processing is as follows:

- One-byte input and output through the standard input/output device
- One-line input and output through the standard input/output device
- Opens and closes a file
- Inputs and outputs one byte from and to a file
- Inputs and outputs a line from and to a file
- Checks the end of file (EOF)
- Moves and/or acquires the current address of the file pointer

The TRAP_ADDRESS command is used to enable this function. Write a subroutine branch instruction (BSR, JSR or BSRF) to a specific address for input or output in the user program. After initiating the simulator/debugger, specify the address using the TRAP_ADDRESS command to execute the program. During executing the instruction of the user program, the simulator/debugger executes I/O processing using the contents of R0 and R1 as parameters after detecting a subroutine call instruction (BSR, JSR or BSRF) to a specified address. After completing I/O processing, simulation restarts from the instruction following the subroutine call instruction. For details, refer to section 4.61, TRAP_ADDRESS.

Note: When a JSR, BSR, or BSRF instruction is used as a system call instruction, the instruction following the system call instruction is executed as a normal instruction, not a slot instruction. Therefore, the instruction placed immediately after the system call instruction (JSR, BSR, or BSRF) must not be one that produces different results depending on whether executed as a normal instruction or as a slot instruction.

2.12 Break Conditions

The simulator/debugger provides the following conditions for interrupting the simulation of a user program during execution.

- Break due to the satisfaction of a break command condition
- Break due to the detection of an error during execution of the user program
- Break due to a trace buffer overflow
- Break due to execution of the SLEEP instruction
- Break due to the [Stop] button

(1) Break Due to the Satisfaction of a Break Command Condition

There are five break commands as follows:

— BREAKPOINT: Break based on the address of the instruction executed

— BREAKACCESS: Break based on access to a range of memory

— BREAKDATA: Break based on the value of data written to memory

— BREAKREGISTER: Break based on the value of data written to a register

— BREAKSEQUENCE: Break based on a specified execution sequence

When a break condition is satisfied during user program execution, the instruction at the breakpoint may or may not be executed before a break depending on the type of break, as listed in table 2.3.

Table 2.3 Processing When a Break Condition is Satisfied

Command	Instruction When a Break Condition is Satisfied
BREAKPOINT	Not executed
BREAKACCESS	Executed
BREAKDATA	Executed
BREAKREGISTER	Executed
BREAKSEQUENCE	Not executed

For BREAKPOINT and BREAK_SEQUENCE, if a breakpoint is specified at an address other than the beginning of the instruction, the break condition will not be detected.

When a break condition is satisfied during user program execution, a break condition satisfaction message is displayed and execution stops.

(2) Break Due to the Detection of an Error During Execution of the User Program

The simulator/debugger detects simulation errors, that is, program errors that cannot be detected by the CPU exception generation functions. The EXEC_MODE command specifies whether to stop or continue the simulation when such an error occurs. Table 2.4 lists the error messages, error causes, and the action of the simulator/debugger in the continue mode.

Table 2.4 Simulation Errors

Error Message	Error Cause	Processing in Continue Mode
Memory Access Error	Access to a memory area that has not been allocated Write to a memory area having the write protect attribute Read from a memory area having the read disable attribute Access to a memory area where memory does not exist	On memory write, nothing is written; on memory read, all bits are read as 1
Illegal Operation	Zero division executed by the DIV1 instruction	Operates similar to the actual device operation
Illegal DSP Operation	Shift of more than 32 bits executed by the PSHA instruction Shift of more than 16 bits executed by the PSHL instruction	Operates similar to the actual device operation
Invalid DSP Instruction Code	Invalid DSP instruction code	Always stops
TLB Multiple Hit	Hit to multiple TLB entries at MMU address translation	Undefined

When a simulation error occurs in the stop mode, the simulator/debugger returns to the command wait state after stopping instruction execution and displaying the error message. Table 2.5 lists the states of the program counter (PC) and status register (SR) at simulation error stop.

Table 2.5 Register States at Simulation Error Stop

Error Message	PC Value	SR Value
Memory Access Error	When an instruction is read: • SH-DSP The third instruction address before the instruction that caused the error. • SH-1, SH-2, SH-3, SH-3E, SH-2E, and SH-4 The instruction address before the instruction that caused the error. The slot address if an error occurs when a branch destination is read. When an instruction is executed: The instruction address following the instruction that caused the error.	Unchanged
Illegal Operation	The instruction address following the instruction that caused the error.	
Illegal DSP Operation	The second instruction address following the instruction that caused the error.	
Invalid DSP Instruction Code	The second instruction address following the instruction that caused the error.	
TLB Multiple Hit	The address of the instruction that caused the error.	

Use the following procedure when debugging programs which include instructions that generate simulation errors.

(a) First execute the program in the stop mode and confirm that there are no errors except those in the intended locations.

(b) After confirming the above, execute the program in the continue mode.

Note: If an error occurs in the stop mode and simulation is continued after changing the simulator mode to the continue mode, simulation may not be performed correctly. When restarting a simulation, always restore the register contents (general, control, and system registers) and the memory contents to the state prior to the occurrence of the error.

(3) Break Due to Trace Buffer Full

With the break mode specified by the TRACE_CONDITION command, the simulator/debugger stops its execution when the trace buffer becomes full during instruction execution, displaying the following message.

```
Trace buffer full
```

(4) Break Due to Execution of the SLEEP Instruction

When the SLEEP instruction is executed during instruction execution, the simulator/debugger stops execution. The following message is displayed when execution is stopped.

```
Sleep
```

Note: When restarting execution, change the PC value to the instruction address at the restart location.

(5) Break Due to the [Stop] Button or Ctrl + C Keys

Users can forcibly terminate execution by pressing the [Stop] button or Ctrl + C keys during instruction execution. The following message is displayed when execution is terminated.

```
User break
```

Execution can be resumed with the GO or STEP command.

2.13 Floating-Point Data

Floating-point numbers can be displayed and input for the following real-number data, which makes floating-point data processing easier.

- Data when the [Break Data] or [Break Register] menu is opened
- Data on the Dump window
- Register value on the Registers window
- Input value in the Registers window

The floating-point data format conforms to the ANSI C standard.

When floating-point data is converted from decimal to binary, the rounding mode of the following two can be selected by the ROUND_MODE command.

- RN mode (Round to Nearest)
- RZ mode (Round to Zero: default)

If a denormalized number is specified for binary-to-decimal or decimal-to-binary conversion, it is converted to zero in the RZ mode, but it is not converted and is left as a denormalized number in the RN mode. If an overflow occurs for decimal-to- binary conversion, the maximum floating point number is specified for the RZ mode and infinity is specified for the RN mode.

Section 3 Operation

This section explains how to start the interface software and check the simulator/debugger operation.

This section assumes that the user knows how to operate the window system on the host computer.

3.1 Setting a Path and Environment Variables

Manually set a path and environment variables when not using the installer to add them to the shell script.

(1) Setting Path: Add the directory of the interface software (csdsh) to the current path specification.

%***setΔpath=($pathΔ<interface software directory path>)(RET)***

(2) Setting Environment Variables: The interface software uses the following environment variables.

— HS_CA_HOM

Specifies the directory of the interface software definition files.

%***setenvΔHS_CA_HOMΔ<definition file directory path> (RET)***

— HS_CA_DEF

Specifies the summary file for the interface software definition files.

%***setenvΔHS_CA_DEFΔ<summary file name> (RET)***

— HS_CA_INT

Specifies a setup file to automatically determine the initial settings during interface software initiation. Specify a setup file on the current directory by referring to the setup file sample on the definition file directory. Refer to section 3.2, Start-up, for details on the setup file contents.

%***setenvΔHS_CA_INTΔ<setup file name> (RET)***

— HS_CA_SIM

Specifies a CPU type to automatically determine the CPU of the interface software. Specify SH1, SH2, SH3, SH4, SH2E, SH3E, or SH-DSP for the CPU type.

%***setenvΔHS_CA_SIMΔ<CPU type> (RET)***

3.2 Start-up

(1) Interface software start-up

The interface software command format is as follows:

% ***csdsh[Δ<setup file name>](RET)***

After start-up, the following message appears.

```
SH SERIES CYCLE-ACCURATE SIMULATOR/DEBUGGER Vn.m
Copyright (C) Hitachi,Ltd.1998
Copyright (C) Hitachi ULSI System Co.,Ltd.1998
Licensed Material of Hitachi,Ltd.
```

(2) Setup file

Used for specifying the simulator, backup file and replay file to be used.

Use the editor to create the setup file. When the setup file is not specified at start-up, the setup window as shown in figure 3.1 appears, in which each item explained below needs to be entered.

Each item is specified in the setup file or on the setup window as follows:

(a) Simulator specifications

The simulator specifying format is as follows (see below for the parameter at start-up):

<simulator/debugger name>[Δ<parameter at start-up>]

SH-1/SH-2 series:

```
sdsh12[Δ-cpu=<CPU information file name>](RET)
  (1)         (2)
```

SH-2E series:

```
sdsh2e[Δ-cpu=<CPU information file name>](RET)
  (1)         (2)
```

SH-DSP series:

```
sdshdsp[Δ-cpu=<CPU information file name>](RET)
  (1)         (2)
```

SH-3/SH-3E series:

```
sdsh3e[Δ-cpu=<CPU information file name>][Δ-endian={big|little}](RET)
  (1)         (2)                            (3)
```

SH-4 series:

```
sdsh4[Δ-cpu=<CPU information file name>][Δ-endian={big|little}](RET)
  (1)         (2)                            (3)
```

Description: (1) The command name of the simulator/debugger program registered in the host computer.

(2) When this option is specified, the simulator/debugger reads CPU information from the specified file to use it as a memory map.

".cpu" is assumed when no file format is specified.

(3) Specifies the endian type (for SH-3, SH-3E, and SH-4 series only)

big: Big endian

little: Little endian

(b) Backup file specifications (omissible)

The backup file, which can be created when quitting the simulator/debugger, is used to save the window position and size as well as the settings on the window. By specifying the backup file at start-up, the settings at the previous quitting can be restored. The backup file is specified as follows:

BAKΔ<backup file name>

(c) Replay file specifications (omissible)

The replay file, which can be created using the recording function of the simulator/debugger, is used to save operation on the simulator/debugger (inputs of the simulator/debugger commands and clicks of the buttons). By specifying the replay file at start-up, default settings and other operation required for debugging can be automatically executed. The replay file is specified as follows:

REPΔ<replay file name>

(d) Example of setup file

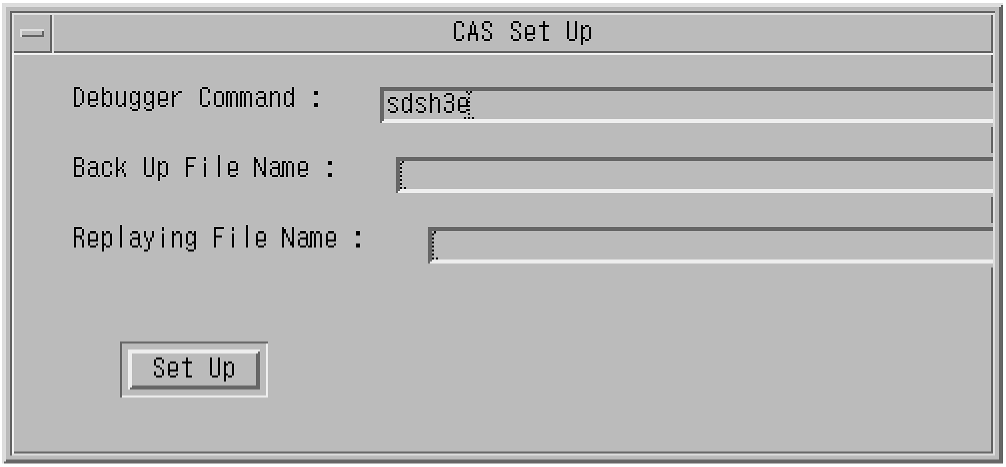
An example of specifying the backup and replay files using the SH-3 simulator/debugger is as follows:

```
# Set up file
sdsh3e
BAK backup
REP recover
```

(3) Set Up window

Displays the items specified in the setup file.

When the setup file is specified with environment variable HS_CA_INT, the contents of the setup file are displayed. When the setup file is specified on the command line, the interface software is started with the contents of the setup file specified on the command line, and this window will not be displayed.



Debugger Command : sdsh3e

Back Up File Name :

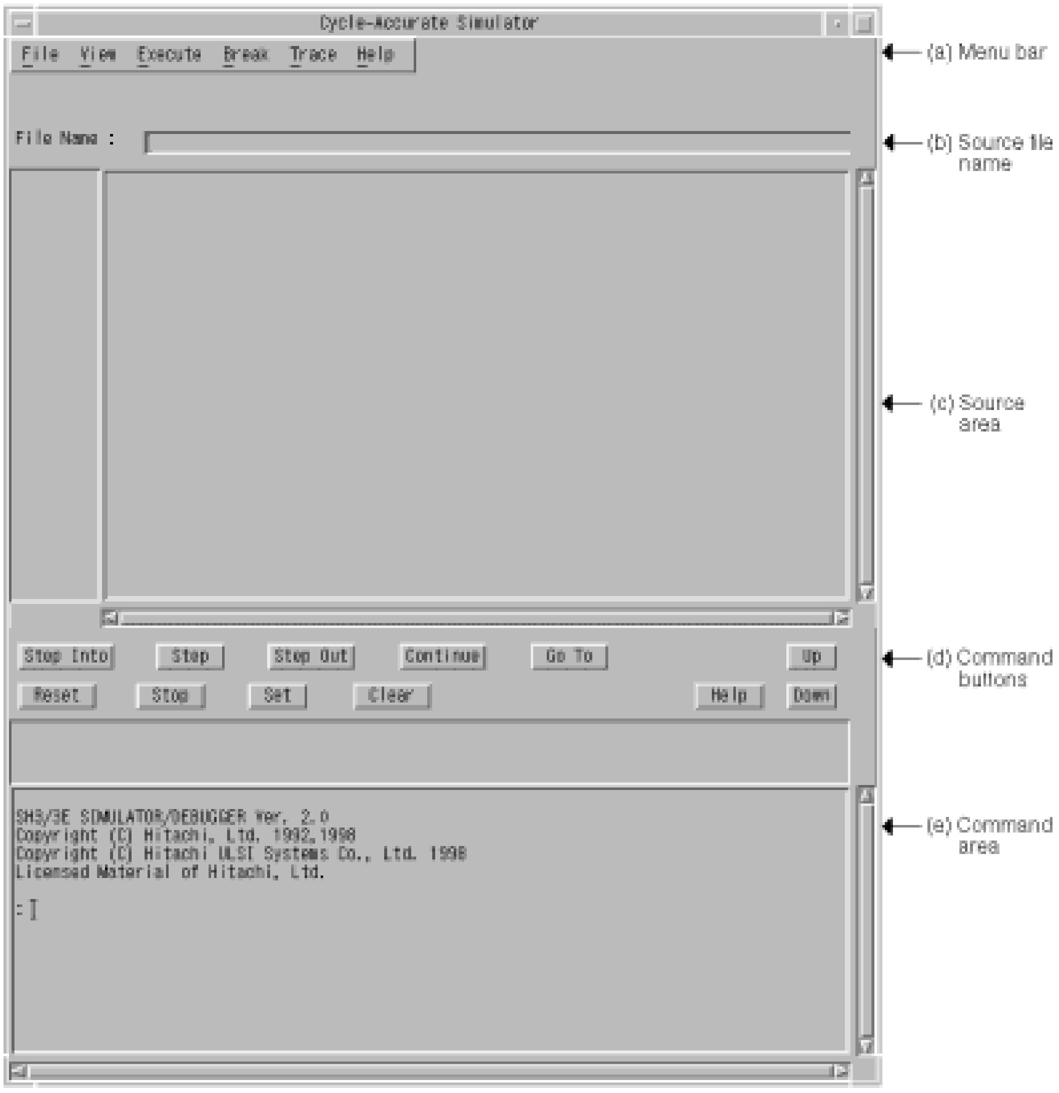
Replaying File Name :

Figure 3.1 Set Up Window

3.3 Windows

(1) Base window

The interface software displays the base window after setup. Operation executed on this window is as follows: source file display, program execution, source level debugging such as break point setting, release, and symbol content display, subwindow selection, and simulator/debugger command input.



```
SH3/3E SIMULATOR/DEBUGGER Ver. 2.0
Copyright (C) Hitachi, Ltd. 1992,1998
Copyright (C) Hitachi ULSI Systems Co., Ltd. 1998
Licensed Material of Hitachi, Ltd.

:
```

Figure 3.2 Base Window

(a) Menu bar

The menu bar has the following items (refer to section 6, Windows and Dialog Boxes, for details):

- File: Selects the subwindow for file-related operation (e.g. loading and saving).
- View: Selects the subwindow for display-related operation such as registers, symbols and disassembly.
- Execute: Selects the subwindow for execution-related operation.
- Break: Selects the subwindow for break-related operation.
- Trace: Selects the subwindow for trace-related operation.
- Help: Selects the subwindow having the help function.

(b) Source file name

Displays the name of the source file displayed in the source area with the absolute path name.

(c) Source area

- Displays the source file contents of the user load module.
- Mark "PC->" shows the line where execution has stopped (value of the program counter).

  The display of the source file contents is updated at execution stop.
- Mark "BP->" shows the line where a breakpoint is set.

(d) Command button

Available command buttons are as follows:

- Step Into

  When a function call exists in the source line indicated by the program counter, executes up to the first line of the called function.
- Step

  Executes the line indicated by the program counter. If a function call exists in the source line, execution stops after executing the whole of the function.
- Step Out

  When the program counter indicates a line in a function, executes the function and stops when returning to the calling function.
- Continue

  Starts execution from the address indicated by the program counter.
- Go To

  Executes to the line selected in the source area (clicked line)
- Reset

  Executes the Reset command of the debugger.
- Stop

  Forcibly stops the running debugger command.

- Set

  Sets a breakpoint on the line selected in the source area (clicked line), displaying the "BP->" mark. If a breakpoint cannot be set on the selected line (no corresponding address found), the address is searched in the direction to the smaller line number to set a breakpoint.

- Clear

  Clears the breakpoint on the line selected in the source area (clicked line), clearing the "BP->" mark.

- Help

  Outputs these descriptions on the help window.

- Up
- Down

  Changes the display ratio (in the vertical direction) of the source and command areas.

  Up: Reduces the source area and expands the command area.

  Down: Expands the source area and reduces the command area.

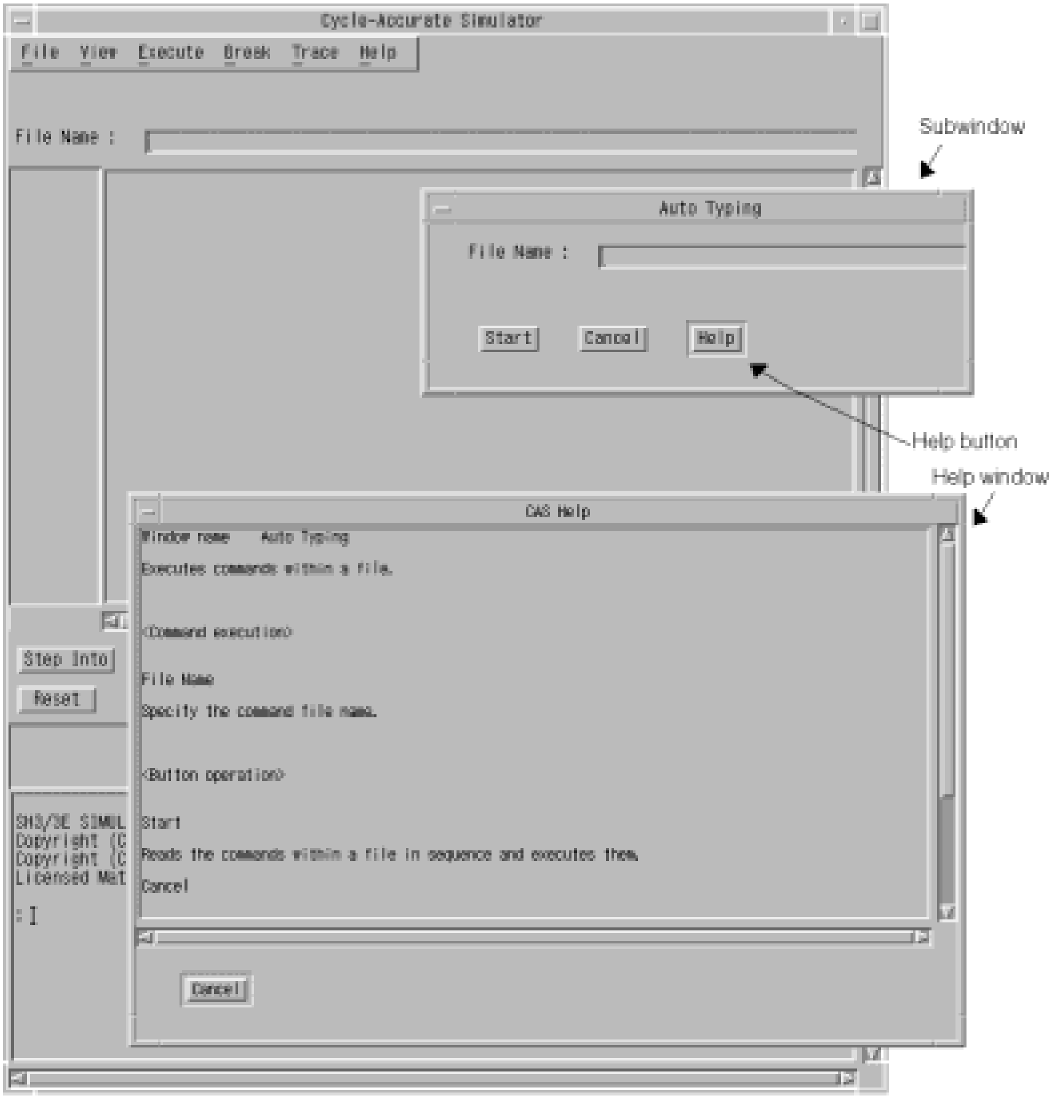
Note: On a load module which has been optimally compiled, the "PC->" mark in the source area may not move as intended in the source file during program execution using the Step, Step Into, Step Out, Continue or Go To button.

(e) Command area

Used to directly input simulator/debugger commands and display the results of command execution. For available commands, refer to section 4, Simulator/Debugger Commands.

(2) Subwindow and help window

The subwindow is opened by pulling the menu button on the base window down and selecting the menu item ("Menu Item..."). Each subwindow also has the Help button to open the help window for displaying its functions.



Window name   Auto Typing

Executes commands within a file.

<Command execution>

File Name

Specify the command file name.

<Button operation>

Start

Reads the commands within a file in sequence and executes them.

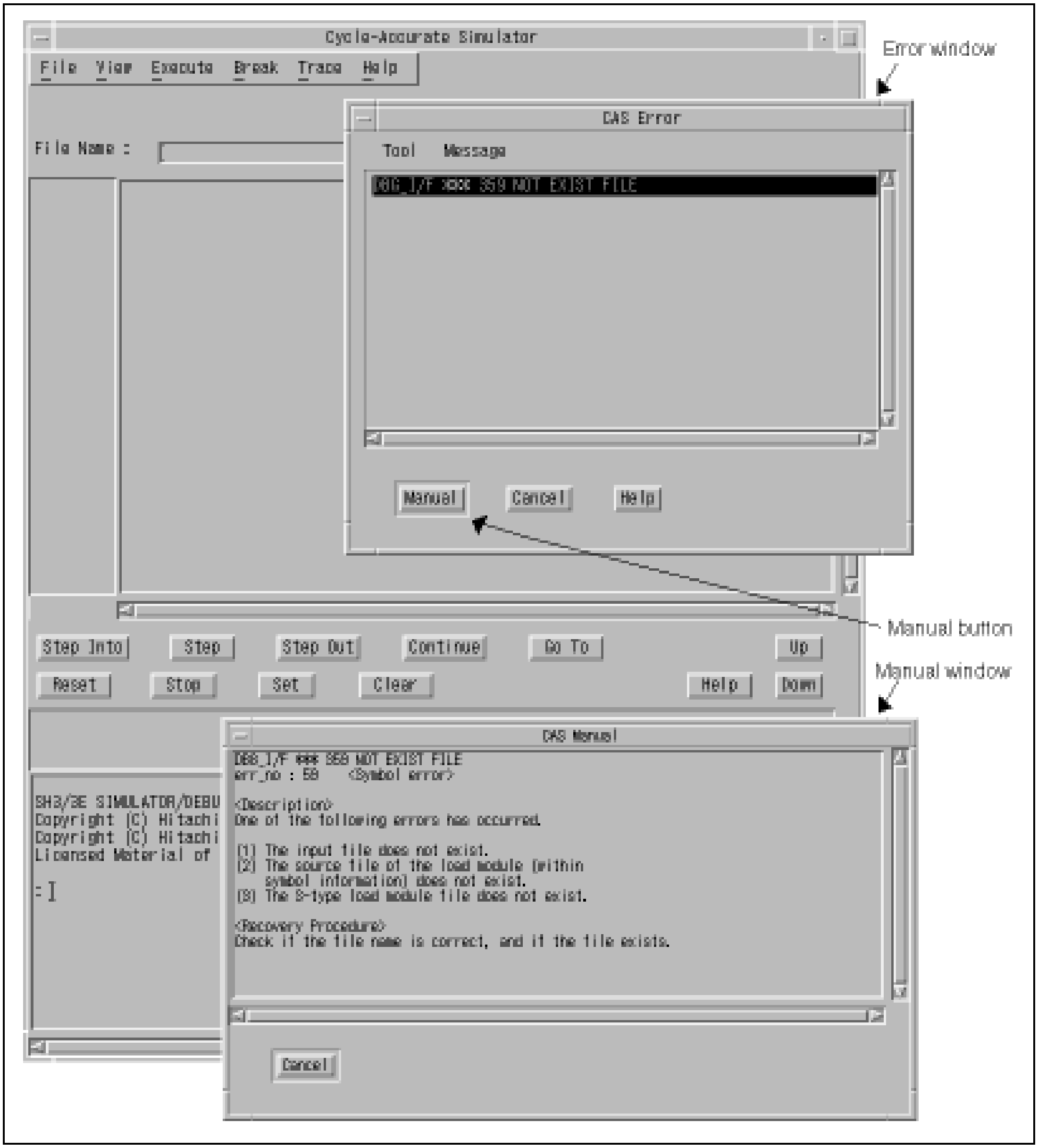
Cancel

```
SH3/3E SIMUL
Copyright (C
Copyright (C
Licensed Mat

:I
```

Figure 3.3   Subwindow and Help Window

(3) Error window and manual window

When an error occurs, the error window is opened to display error messages. Using the Manual button on the error window, the manual window for explaining error messages can be opened. When there are several messages, select the one you want to view on the error window and then press the Manual button.



```
DBG_I/F *** 359 NOT EXIST FILE
```

```
SH3/3E SIMULATOR/DEBU
Copyright (C) Hitachi
Copyright (C) Hitachi
Licensed Material of

: ]
```

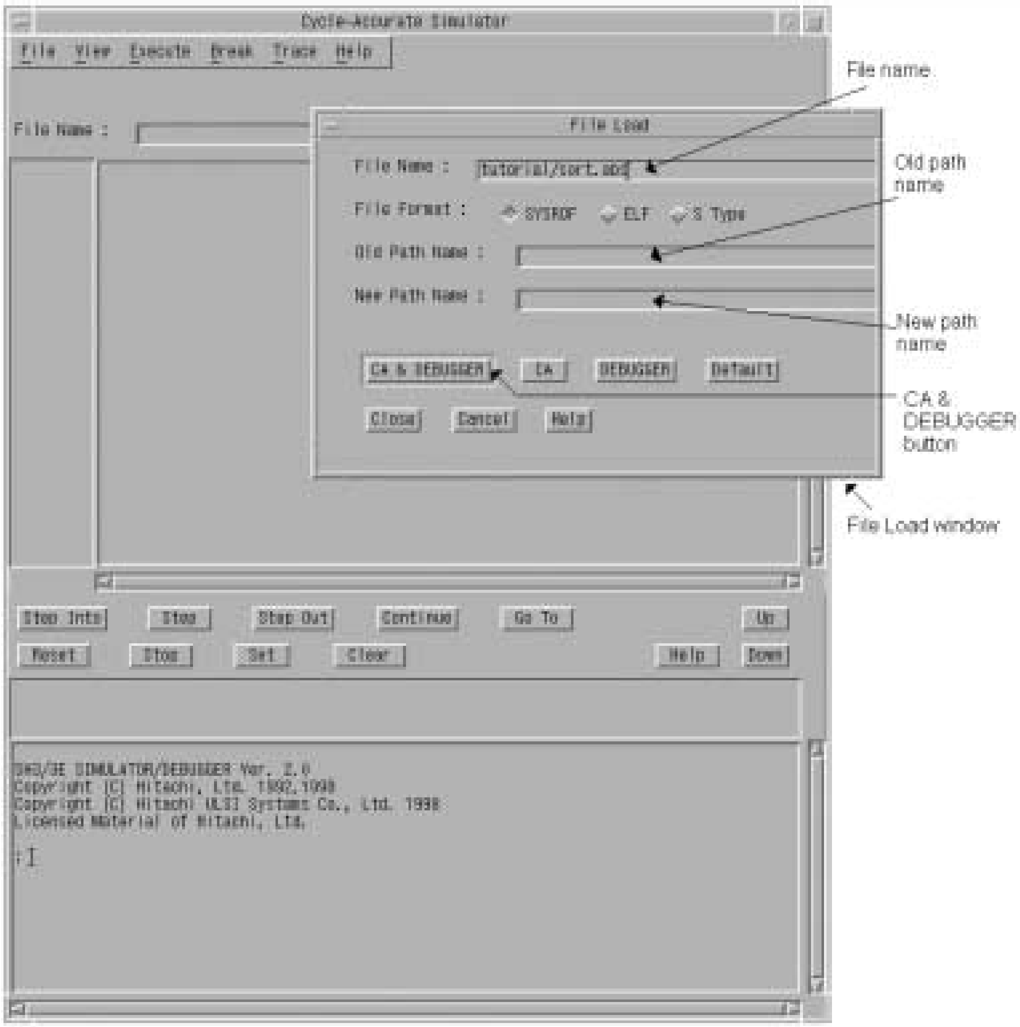
```
DBG_I/F *** 359 NOT EXIST FILE
err_no : 58    <Symbol error>

<Description>
One of the following errors has occurred.

[1] The input file does not exist.
[2] The source file of the load module [within
    symbol information] does not exist.
[3] The S-type load module file does not exist.

<Recovery Procedure>
Check if the file name is correct, and if the file exists.
```

Figure 3.4 Error Window and Manual Window

3.4 Loading Load Module

Use the File Load window to load a load module. Input the file name output by the inter-module optimizer and press the CA&DEBUGGER button, which automatically triggers the simulator/debugger to allocate load module memory.

When the source file has been moved from the directory where the source file was stored when the load module was output by the inter-module optimizer, enter the path for the old source file directory in the Old Path Name in the File Load window, and enter the path for the current source file directory in the New Path Name.



```
SH3/3E SIMULATOR/DEBUGGER Ver. 2.0
Copyright (C) Hitachi, Ltd. 1992,1998
Copyright (C) Hitachi ULSI Systems Co., Ltd. 1998
Licensed Material of Hitachi, Ltd.
:I
```

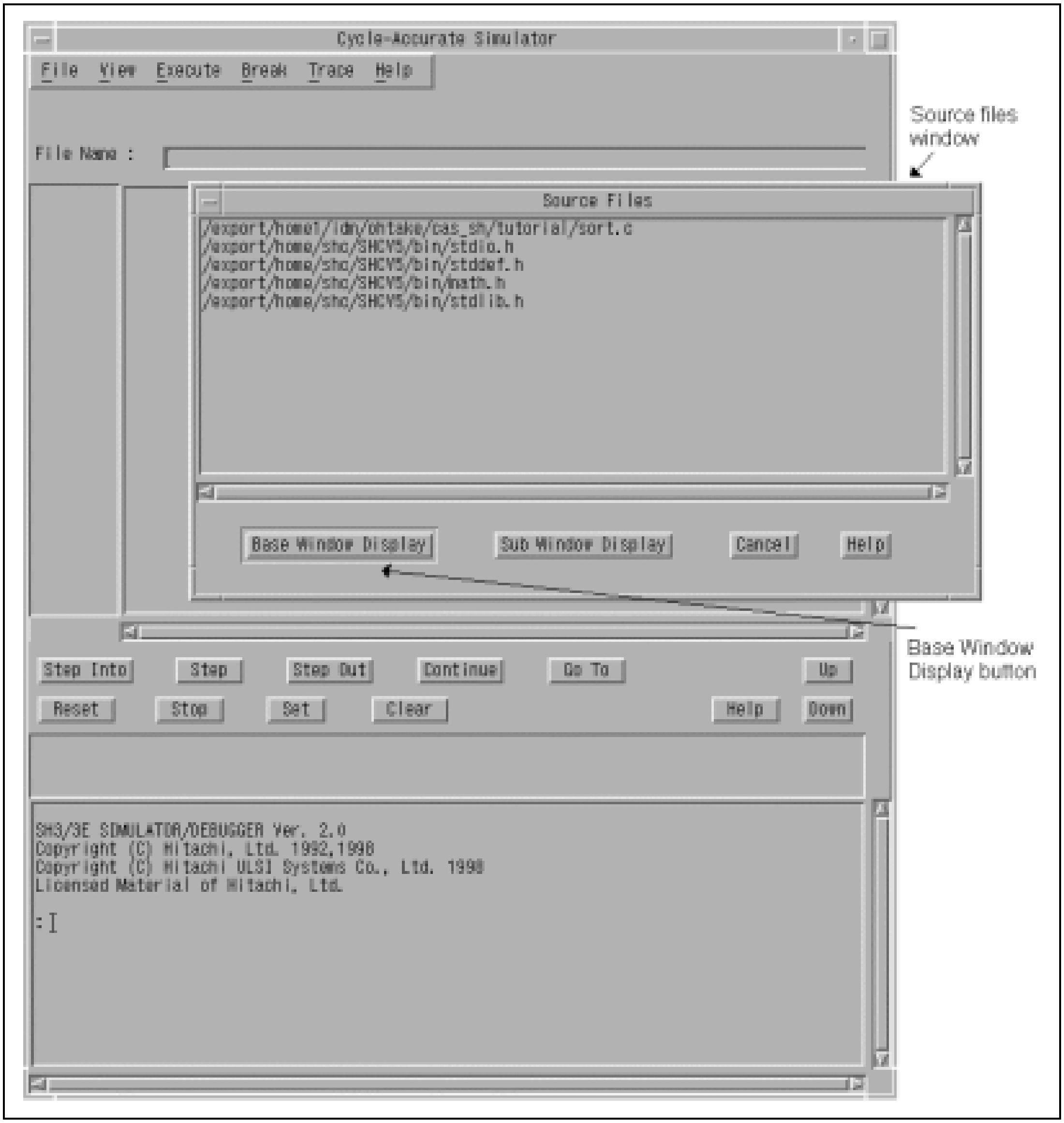
Figure 3.5   Example of Load Module Selection

3.5 Displaying Source File

The source file can be displayed or the file to display can be changed by the following procedure.

(a) After loading is completed, open the Source Files window or Function Name window.

(b) Select the Source Files or Function Name to display and press the Base Window Display button.



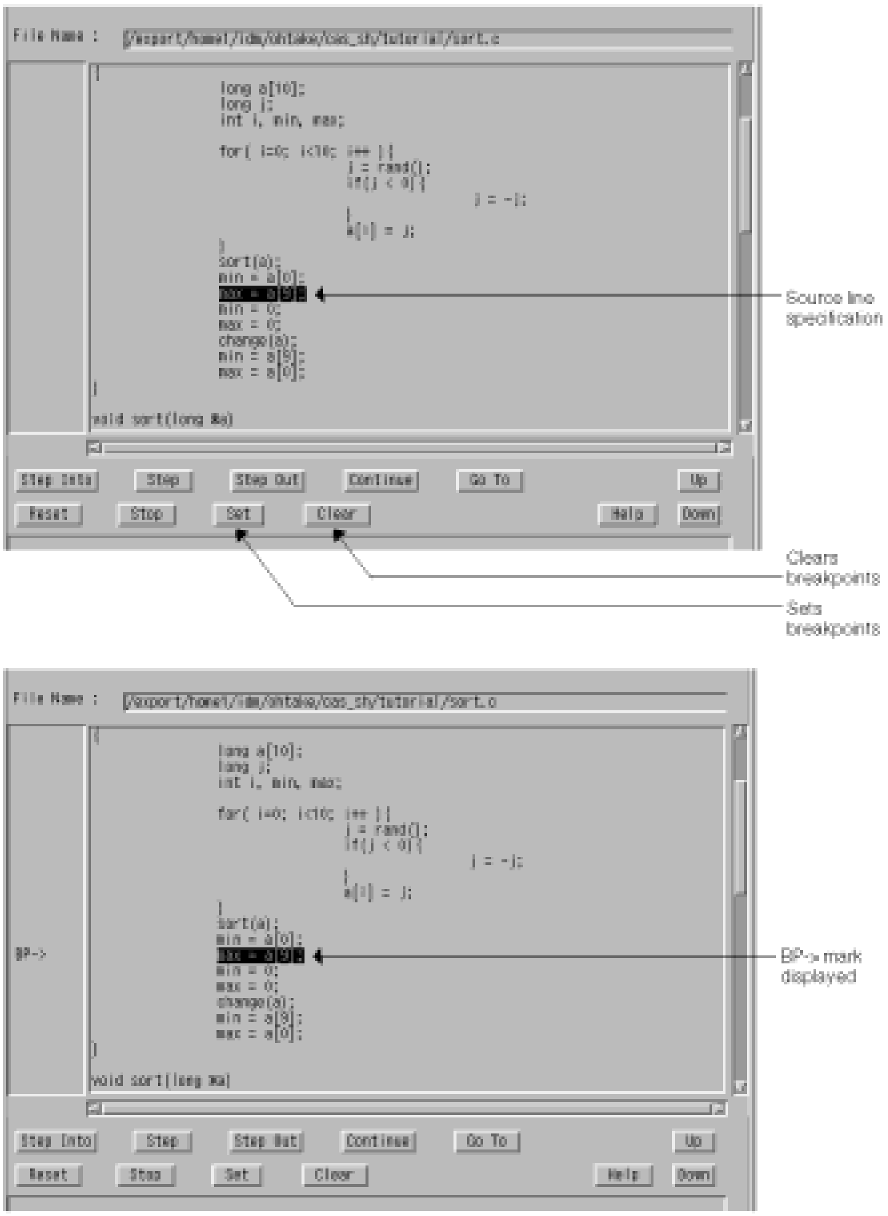
```
/export/home1/idn/ohtake/cas_sh/tutorial/sort.c
/export/home/shc/SHCV5/bin/stdio.h
/export/home/shc/SHCV5/bin/stddef.h
/export/home/shc/SHCV5/bin/math.h
/export/home/shc/SHCV5/bin/stdlib.h
```

```
SH3/3E SIMULATOR/DEBUGGER Ver. 2.0
Copyright (C) Hitachi, Ltd. 1992,1998
Copyright (C) Hitachi ULSI Systems Co., Ltd. 1998
Licensed Material of Hitachi, Ltd.

:
```

Figure 3.6 Example of Source File Selection

3.6 Setting Breakpoints

A breakpoint can be set by specifying the source line in the source area and pressing the Set button. To clear the breakpoint, specify the source line and press the Clear button.



/export/home1/idm/ohtake/cas_sh/tutorial/sort.c

```
{
                long a[10];
                long j;
                int i, min, max;

                for( i=0; i<10; i++ ){
                                j = rand();
                                if(j < 0){
                                                j = -j;
                                }
                                a[i] = j;
                }
                sort(a);
                min = a[0];
                max = a[9];
                min = 0;
                max = 0;
                change(a);
                min = a[9];
                max = a[0];
}

void sort(long *a)
```

/export/home1/idm/ohtake/cas_sh/tutorial/sort.c

```
{
                long a[10];
                long j;
                int i, min, max;

                for( i=0; i<10; i++ ){
                                j = rand();
                                if(j < 0){
                                                j = -j;
                                }
                                a[i] = j;
                }
                sort(a);
                min = a[0];
BP->            max = a[9];
                min = 0;
                max = 0;
                change(a);
                min = a[9];
                max = a[0];
}

void sort(long *a)
```

Figure 3.7 Example of Setting Breakpoint

3.7 Specifying Symbolic Debugging for Addresses

Addresses can be symbolically input using line numbers, function names or variable names instead of numeric values (see figure 3.8).

This section explains how to specify symbols. <unit name> is a character string excluding ".<suffix>" from the object module file name output by the compiler or the assembler. When <unit name> and <file name> include no special characters (other characters than alphanumerics, "_" or $), double quotation mark (") can be omitted.

(1) Line number

```
[& <unit name>]% <file name> #<line number>
```

Specify <unit name> when the specified source file is included in several units.

(2) Function name

```
[& <unit name>] |<function name>
```

Specify <unit name> when the specified function name is used in several units as an internal function.

(3) Variable name

For variable names, variable and constant names of C language and label and EQU names of the assembly language can be specified. For EQU names, values are assumed as addresses. When specifying a structure or union member of the C language, input it in the format of <structure name. member name>.

(a) Externally defined variable

```
!<variable name>
```

(b) Variable inside a unit

```
&" <unit name>" !<variable name>
```

(c) Variable name inside a function

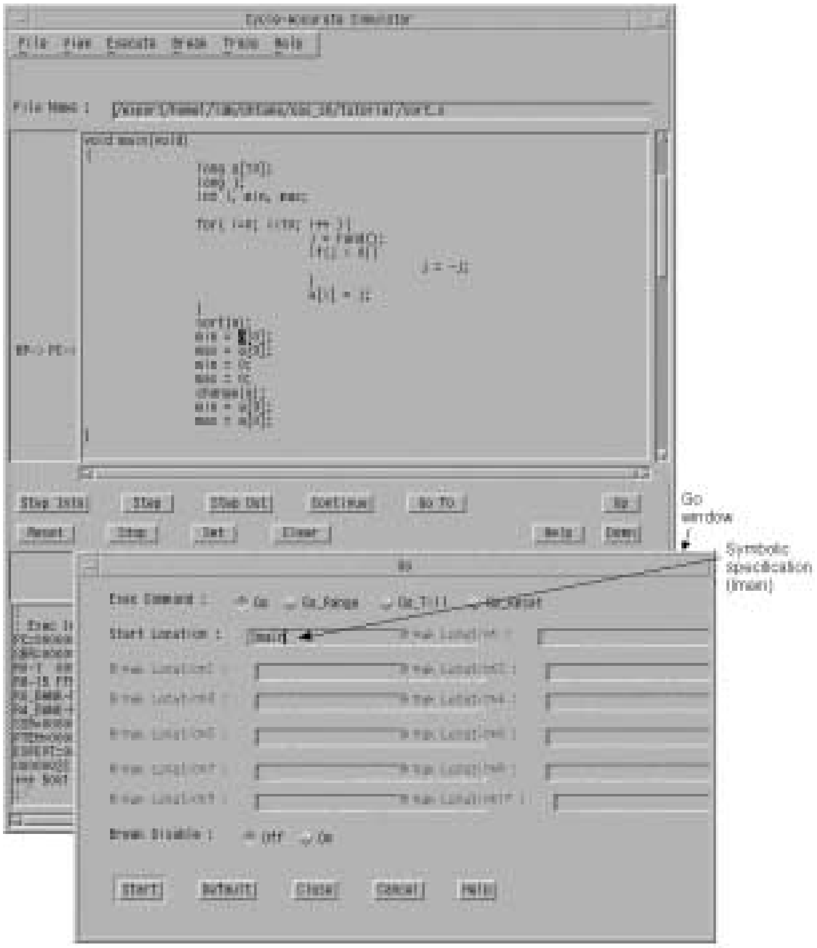
```
[&" <unit name>"] |<function name>!<variable name>
```

Specify <unit name> when the specified function name is used in several units as an internal function.

3.8 Executing Program

Programs can be executed by specifying the program counter (PC) and the stack pointer (SP) on the register window or in the command area and pressing the command button (Step Into, Step, Step Out, Continue, or Go To).

Execution can also be started by specifying the execution start address on the Go window.



```
void main(void)
{
        long a[10];
        long j;
        int i, min, max;

        for( i=0; i<10; i++ ){
                j = rand();
                if( j < 0 ){
                                j = -j;
                }
                a[i] = j;
        }
        sort(a);
        min = a[0];
        max = a[9];
        min = 0;
        max = 0;
        change(a);
        min = a[0];
        max = a[9];
}
```

Exec Command : [✓ Go] Go_Range Go_Till [illegible]

Start Location : main

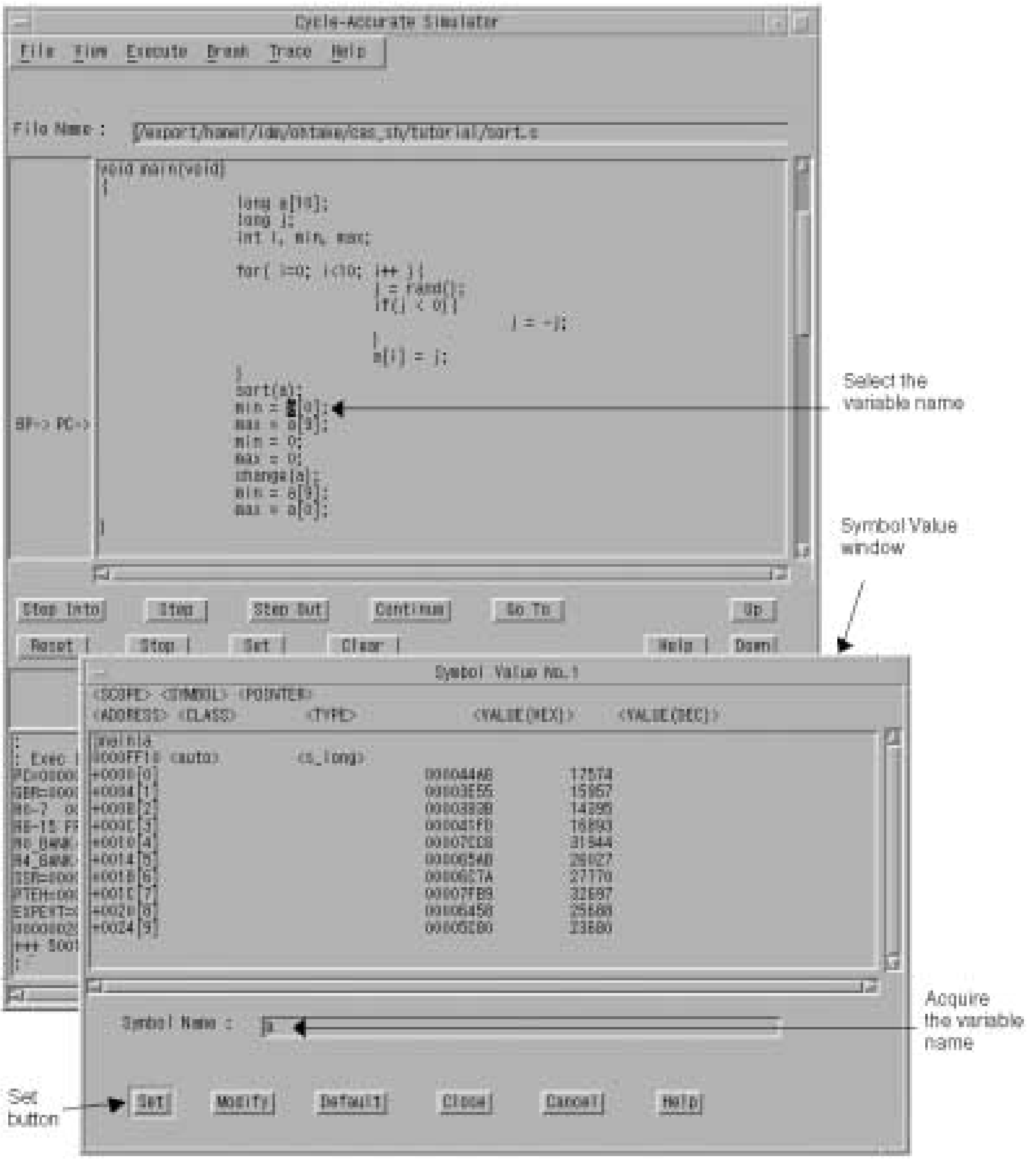
Break Disable : [✓ Off] On

Figure 3.8 Example of Input on Execution Window

3.9 Displaying Variable Contents

The contents of variables can be displayed by the following procedure.

(a) Select the variable name in the source area. (In figure 3.9, variable a is selected.)
(b) Open the Symbol Value window using the View menu, acquiring the variable name.
(c) Press the Set button on the Symbol Value window.



File Name : /export/home1/idm/ohtake/cas_sh/tutorial/sort.c

```
void main(void)
{
                long a[10];
                long j;
                int i, min, max;

                for( i=0; i<10; i++ ){
                        j = rand();
                        if(j < 0){
                                j = -j;
                        }
                        a[i] = j;
                }
                sort(a);
BP-> PC->       min = a[0];
                max = a[9];
                min = 0;
                max = 0;
                change(a);
                min = a[9];
                max = a[0];
}
```

```
<SCOPE> <SYMBOL> <POINTER>
<ADDRESS> <CLASS>        <TYPE>              <VALUE(HEX)>    <VALUE(DEC)>
main|a
0000FF10 <auto>          <s_long>
+0000[0]                                     000044A6        17574
+0004[1]                                     00003E55        15957
+0008[2]                                     000037D7        14295
+000C[3]                                     000041FD        16893
+0010[4]                                     00007CC8        31944
+0014[5]                                     000065AB        26027
+0018[6]                                     00006C7A        27770
+001C[7]                                     00007FB9        32697
+0020[8]                                     00006458        25688
+0024[9]                                     00005C80        23680
```

Symbol Name : a

Figure 3.9 Example of Variable Content Display

3.10 Analyzing Execution Performance

Execution performance can be analyzed by the following procedure.

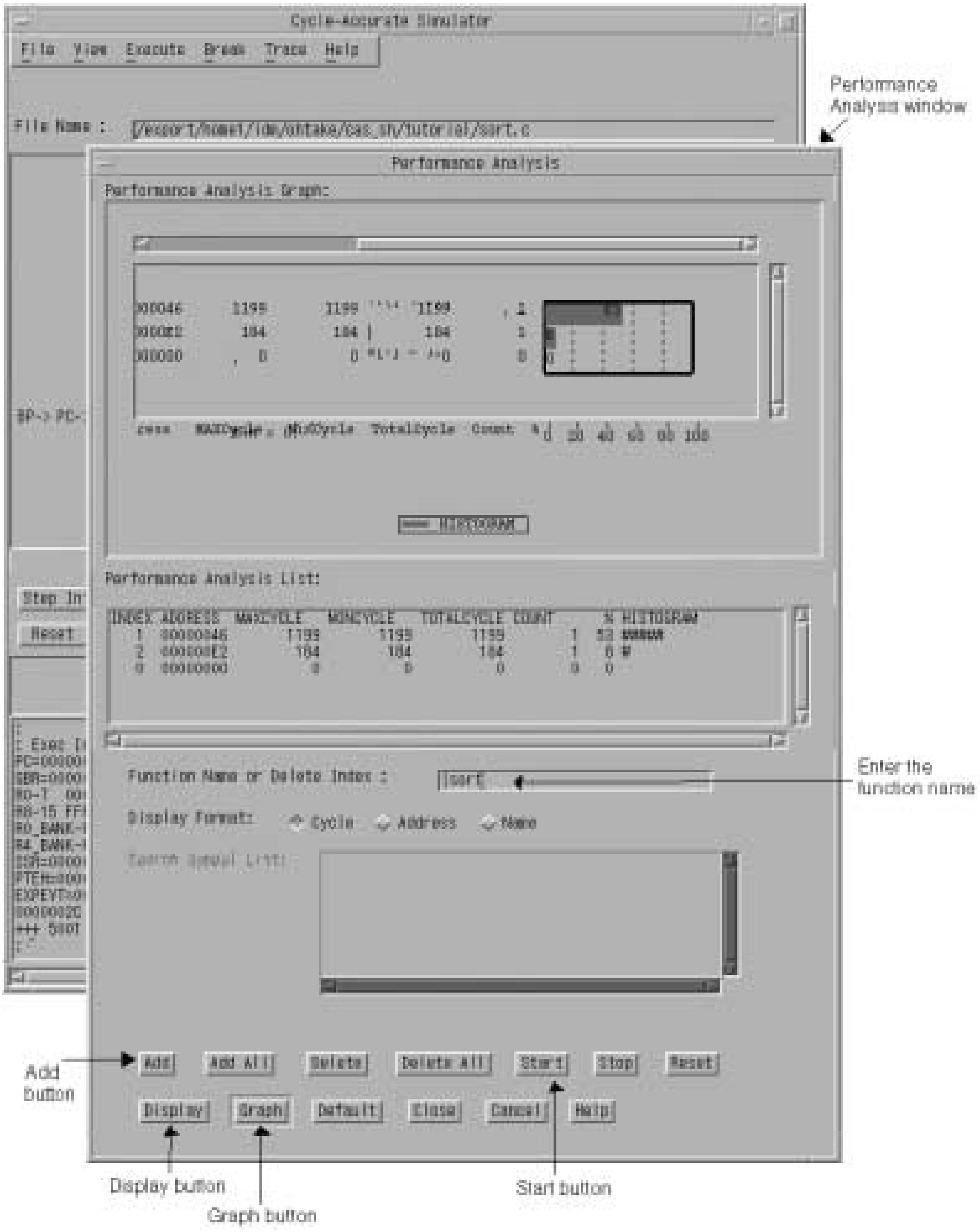
(a) Open the Performance Analysis window using the View menu.

(b) Input the function name to analyze in the Function Name or Delete Index column and press the Add button.

(c) Press the Start button to start analysis.

(d) Press the Display button to display the analysis results.

(e) Press the Graph button to display the results in a graph.



File Name : /export/home1/idm/ohtake/cas_sh/tutorial/sort.c

```
INDEX ADDRESS   MAXCYCLE   MINCYCLE   TOTALCYCLE COUNT      % HISTOGRAM
    1 00000046      1199       1199         1199     1   53 ######
    2 000000E2       104        104          104     1    0 #
    0 00000000         0          0            0     0    0
```

Function Name or Delete Index : sort

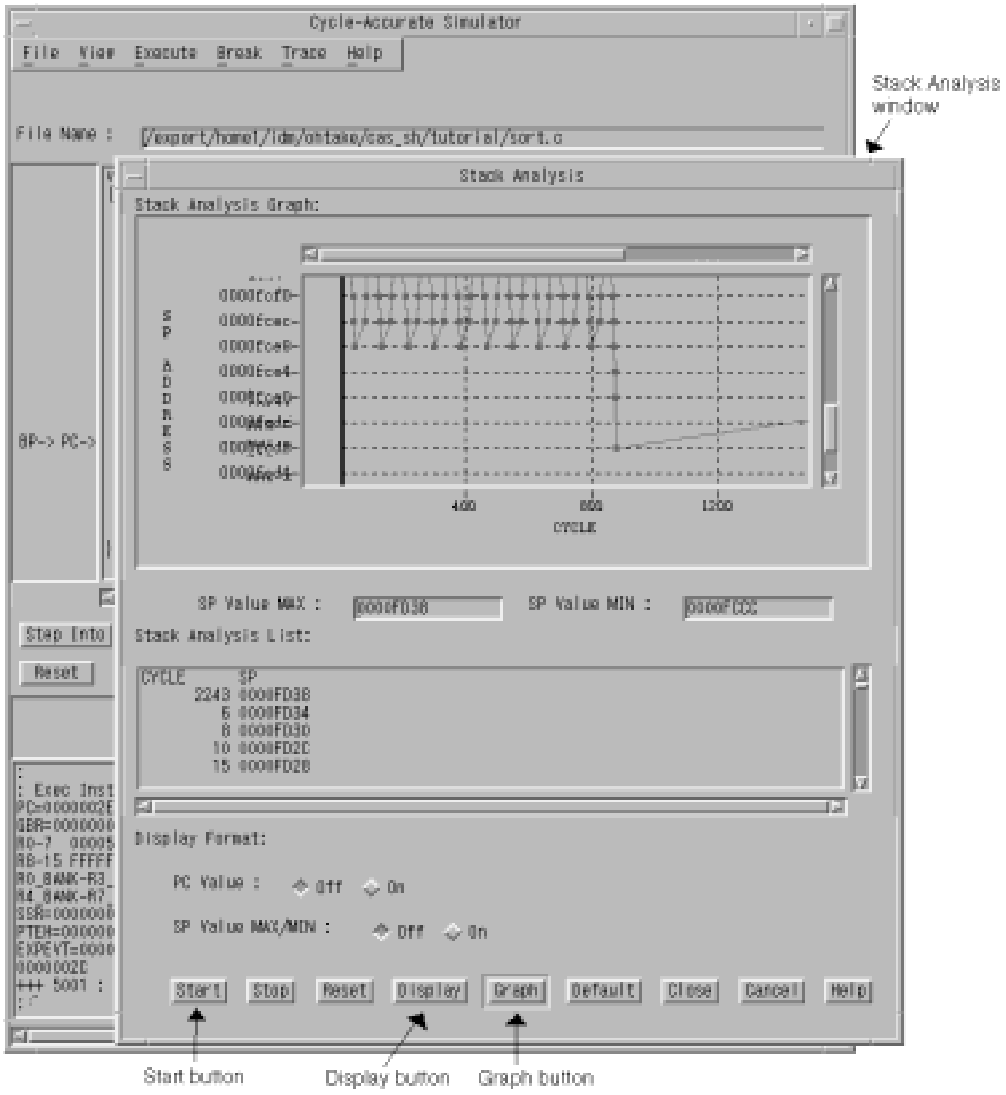
Figure 3.10   Example of Performance Display

RENESAS

3.11 Analyzing Stack Use Status

The stack use status can be analyzed by the following procedure.

(a) Open the Stack Analysis window using the View menu.
(b) Press the Start button to start analyzing the stack use status.
(c) Press the Display button to display the analysis results.
(d) Press the Graph button to display the results in a graph.



```
SP Value MAX : 0000FD38      SP Value MIN : 0000FCCC
Stack Analysis List:
CYCLE      SP
 2243 0000FD38
    6 0000FD34
    8 0000FD30
   10 0000FD2C
   15 0000FD28
Display Format:
PC Value :         Off   On
SP Value MAX/MIN :  Off   On
Start  Stop  Reset  Display  Graph  Default  Close  Cancel  Help
```

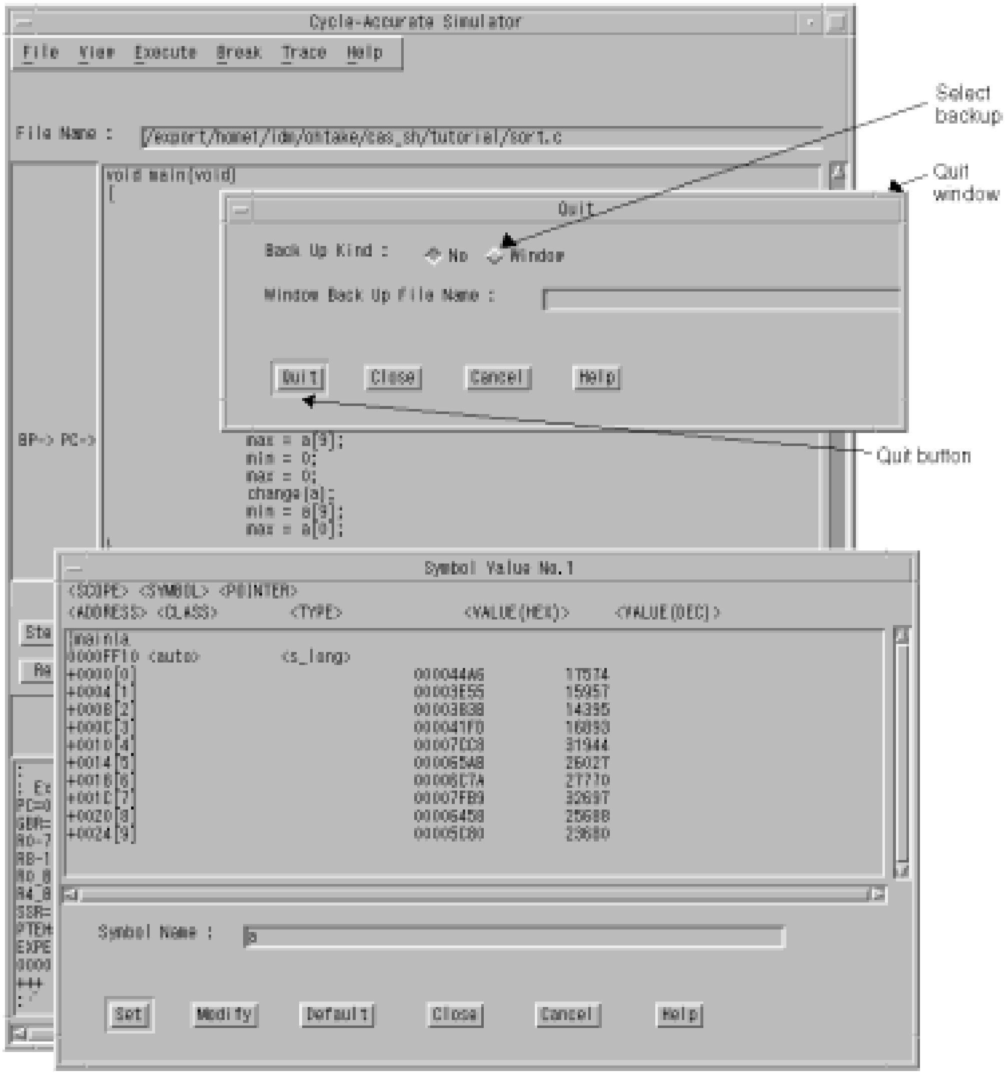
Figure 3.11 Example of Stack Trace Display

3.12 Quit

The Quit window is displayed by selecting the Quit button on the File menu and operation is stopped by pressing the Quit button on the Quit window.

After selecting "Window" for backup selection, the interface software window position and size and setting information on the window can be saved.

The default setting of the file name to save setting information is "HS_CA.BAK".



File Name : /export/home1/idm/ohtake/cas_sh/tutorial/sort.c

```
void main(void)
{
      max = a[9];
      min = 0;
      max = 0;
      change(a);
      min = a[9];
      max = a[0];
}
```

```
<SCOPE> <SYMBOL> <POINTER>
<ADDRESS> <CLASS>          <TYPE>              <VALUE(HEX)>     <VALUE(DEC)>
main\a
0000FF10 <auto>           <s_long>
+0000[0]                                      000044A6          17574
+0004[1]                                      00003E55          15957
+0008[2]                                      0000383B          14395
+000C[3]                                      000041F0          16880
+0010[4]                                      00007CC8          31944
+0014[5]                                      000065AB          26027
+0018[6]                                      00006C7A          27770
+001C[7]                                      00007FB9          32697
+0020[8]                                      00006458          25688
+0024[9]                                      00005C80          23680
```

Symbol Name : a

Figure 3.12 Example of Input on Quit Window

Section 4 Simulator/Debugger Commands

Table 4.1 shows simulator/debugger commands available on the command line.

Table 4.1 Simulator/Debugger Command List

No.	Command	Abbr.	Function
1	ASSEMBLE	AS	Assembles line by line
2	BREAK_CLEAR	BC	Clears breakpoints
3	BREAK_ENABLE	BE	Enables or disables breakpoints
4	BREAKACCESS	BA	Sets break conditions based on access to a memory range
5	BREAKACCESS_DISPLAY	BAD	Displays break conditions based on access to a memory range
6	BREAKDATA	BD	Sets a break condition based on the value of memory data
7	BREAKDATA_DISPLAY	BDD	Displays the break conditions based on the value of memory data
8	BREAKPOINT	BP	Sets breakpoints based on the address of instruction execution
9	BREAKPOINT_DISPLAY	BPD	Displays breakpoints based on the address of instruction execution
10	BREAKREGISTER	BR	Sets break conditions based on the value of data in a register
11	BREAKREGISTER_DISPLAY	BRD	Displays break conditions based on the value of data in a register
12	BREAKSEQUENCE	BS	Sets breakpoints with execution sequence specified
13	BREAKSEQUENCE_DISPLAY	BSD	Displays breakpoints with execution sequence specified
14	COMPARE	CMP	Compares memory contents
15	DATA_SEARCH	DS	Searches for data
16	DISASSEMBLE	DA	Disassembles and displays memory contents
17	DISPLAY_CHARACTERS	DCH	Displays character string
18	EXEC_MODE	EM	Switches execution mode
19	FILE_LOAD	FL	Loads file
20	FILE_SAVE	FS	Saves memory data to a file
21	GO	G	Executes instructions continuously

Table 4.1 Simulator/Debugger Command List (cont)

No.	Command	Abbr.	Function
22	GO_RANGE	GR	Executes instructions continuously (with range specified)
23	GO_RESET	GS	Executes user program from the vector address
24	GO_TILL	GT	Executes instructions continuously (with stop address specified)
25	HELP	HE	Displays command names and input formats
26	LOAD_STATUS	LS	Restores the simulator/debugger memory and register status
27	LOG	LO	Starts creating an execution history file
28	LOG_ENABLE	LE	Enables/disables execution history file creation
29	LOG_STOP	LT	Stops creating an execution history file
30	MAP_CLEAR	MC	Clears memory areas
31	MAP_DISPLAY	MI	Displays memory areas
32	MAP_SET	MS	Sets memory areas
33	MEMORY_DISPLAY	MD	Displays memory contents
34	MEMORY_EDIT	ME	Modifies memory contents
35	MEMORY_FILL	MF	Initializes memory areas
36	MEMORY_MOVE	MV	Moves memory blocks
37	PERFORMANCE_ANALYSIS	PA	Sets execution performance analysis
38	PERFORMANCE_ANALYSIS_CLEAR	PC	Clears execution performance analysis
39	PERFORMANCE_ANALYSIS_DISPLAY	PD	Displays execution performance analysis results
40	PERFORMANCE_ANALYSIS_ENABLE	PE	Enables/disables or resets execution performance analysis
41	QUIT	Q	Exits the simulator/debugger
42	RADIX	RX	Sets the radix
43	REGISTER	R	Displays registers
44	RESET	RS	Resets the simulator/debugger
45	ROUND_MODE	RM	Specifies and displays the floating-point rounding mode
46	SAVE_STATUS	SS	Saves the current simulator/debugger status in a file

Table 4.1 Simulator/Debugger Command List (cont)

No.	Command	Abbr.	Function
47	STACK_ANALYSIS	SA	Enables/disables or resets stack use analysis
48	STACK_ANALYSIS_DISPLAY	SD	Displays the stack use analysis results
49	STATUS	ST	Displays the simulator/debugger status
50	STEP	S	Performs step execution (executes subroutine as one step)
51	STEP_G	SG	Specifies step execution address range executing subroutine as one step
52	STEP_INTO	SI	Performs step execution
53	STEP_INTO_G	SIG	Specifies step execution range
54	TLB	TLB	Modifies the TLB contents
55	TLB_DUMP	TLBD	Displays the TLB contents
56	TLB_FLUSH	TLBF	Flushes the TLB contents
57	TLB_SEARCH	TLBS	Searches for the TLB contents
58	TRACE	T	Displays trace buffer
59	TRACE_CONDITION	TC	Sets trace condition, and starts or stops trace
60	TRACE_CLEAR	TL	Clears trace buffer
61	TRAP_ADDRESS	TA	Sets the system call start address
62	TRAP_ADDRESS_DISPLAY	TD	Displays the system call start address
63	TRAP_ADDRESS_ENABLE	TE	Enables/disables the system call start address
64	.<register>	.	Changes the contents of registers

This section explains the command format.

(1)	(3)
(2)	

Format (4)

Parameter (5)

Function (6)

Description (7)

Note (8)

Example (9)

The numbered items in the above format are described below.

(1) Command name.
(2) Command abbreviation.
(3) Command function.
(4) Input format for the command.
(5) Description of command parameters and options.
(6) Command function.
(7) Description on command usage.
(8) Notes on command usage.
(9) Usage examples.

4.1 ASSEMBLE

ASSEMBLE	Assembles line by line
AS	

Format ASSEMBLEΔ<start address> (RET)

Parameter • <start address> Indicates the address to store the results of assembly.

Function This command converts assembly language notations input in interactive mode to machine language in line units and stores the results starting at the indicated start address. In assemble mode, entering "." terminates the AS command, entering "^" moves the position backward by 1 byte, and pressing the Enter key moves the position forward by 1 byte. For descriptions in the assembly language, refer to the SuperH™ RISC engine Cross Assembler User's Manual.

Example To interactively input assembly language notation, convert them to machine language, and store them starting at address H'400:

```
: ASSEMBLE 400 (RET)
00000400: (RET)
00000401: (RET)
00000402: MOV #H'02E,R1(RET)
00000402 MOV #H'02E,R1
00000404: ADD R1,R2(RET)
00000404 ADD R1,R2
00000406: ^(RET)
00000405: .(RET)
:
```

4.2 BREAK_CLEAR

BREAK_CLEAR	Clears breakpoints
BC	

Format BREAK_CLEAR[Δ<index>](RET)

Parameter • <index> Specifies the break number (break number can be checked by breakpoint display).
Unless specified, all breakpoints are cleared.

Function Clears the breakpoint having the specified break number.
The breakpoints set by the following commands can be cleared.
BREAKACCESS, BREAKDATA, BREAKPOINT, BREAKREGISTER, and BREAKSEQUENCE

Example To clear the first breakpoint:

```
: BREAK_CLEAR 0 (RET)
:
```

4.3 BREAK_ENABLE

BREAK_ENABLE	Enables or disables breakpoints
BE	

Format BREAK_ENABLEΔ{E|D}[Δ<index>](RET)

Parameters • Enable/disable {E|D}

E (enable): Enables the set break conditions.

D (disable): Disables the set break conditions.

• <index> Specifies the break number (break number can be checked by breakpoint display).

Unless specified, all breakpoints are enabled or disabled.

Function Enables or disables the breakpoints having the specified break numbers.

The breakpoints set by the following commands can be enabled or disabled.

BREAKACCESS, BREAKDATA, BREAKPOINT, BREAKREGISTER, and BREAKSEQUENCE

Examples 1. To disable the first breakpoint:

```
: BREAK_ENABLE D 0 (RET)
:
```

2. To enable all breakpoints:

```
: BREAK_ENABLE E (RET)
:
```

4.4 BREAKACCESS

BREAKACCESS	Sets break conditions based on access to a memory range
BA	

Format BREAKACCESSΔ<start address>[Δ<end address>][Δ{R|W|RW}](RET)

Parameters
- <start address>[Δ<end address>]
  Specifies the start address or the range of memory for which the simulator/debugger will stop if accessed by the user program.
  When the end address is not specified, the range consists of only the specified address.
- Access type {R|W|RW}

R (read):	Breaks on a read from the specified memory.
W (write):	Breaks on a write to the specified memory.
RW (read/write):	Breaks on either a read from or a write to the specified memory (default).

Function Sets break conditions based on access to memory.
Execution stops when the specified memory range has been accessed.
Up to two memory ranges can be specified.
Note that breakpoints are automatically enabled when a breakpoint is set.

Example To set a breakpoint so that execution stops when the range from address H'1000 to address H'1100 has been read or written:

```
: BREAKACCESS 1000 1100 RW (RET)
:
```

4.5 BREAKACCESS_DISPLAY

BREAKACCESS_DISPLAY	Displays break conditions based on access to a memory range
BAD	

Format BREAKACCESS_DISPLAY (RET)

Function Displays the break conditions based on access to memory in the following format:

<INDEX>:	Break No.
<E/D>:	Enable/disable
<START>:	Start address
<END>:	End address
<ATTR>:	Access type

Example To display the current settings (address is displayed in hexadecimal):

```
: BREAKACCESS_DISPLAY(RET)
  <INDEX> <E/D> <START>  <END>    <ATTR>
    001     E   00001000 00001100   RW
```

4.6 BREAKDATA

BREAKDATA	Sets a break condition based on the value of memory data
BD	

Format BREAKDATAΔ<break address>Δ<data>[;<size>][Δ<option>](RET)

Parameters

- <break address>
  Specifies the address whose contents are to be checked during execution.
- <data>
  Specifies the break condition data.
- <size> Data size {B|W|L|D|S}

B (byte):	Byte data
W (word):	Word data
L (long):	Long-word data (default)
D (double float):	Double-precision floating-point data
S (single float):	Single-precision floating-point data

- <option> Data equal/not equal {EQ|NE}

EQ (equal):	Breaks when the data are equal. (default)
NE (not equal):	Breaks when the data are not equal.

Function This command sets a breakpoint based on data written to memory.
Program execution stops when the break condition is satisfied.
Up to eight breakpoints can be set.
Note that breakpoints are automatically enabled when a breakpoint is set.

Examples (1) To set a breakpoint so that execution stops when word-size data with the value 10 is written to address H'2000:

```
: BREAKDATA 2000 10;W (RET)
:
```

(2) To set a breakpoint so that execution stops when address H'AF00 is changed to byte-size data with a value other than 20:

```
: BREAKDATA 0AF00 20;B NE (RET)
:
```

4.7 BREAKDATA_DISPLAY

BREAKDATA_DISPLAY	Displays the break conditions based on the value of memory data
BDD	

Format BREAKDATA_DISPLAY(RET)

Function Displays the breakpoints based on the value of memory data in the following format:

<INDEX>:	Break No.
<E/D>:	Enable/disable
<ADDRESS>:	Break address
<DATA>:	Written data and data size
<EQ/NE>:	Data equal/not equal

Example To display the currently set breakpoints (note that addresses, and data are displayed in hexadecimal):

```
: BREAKDATA_DISPLAY (RET)
  <INDEX> <E/D> <ADDRESS>          <DATA>                <EQ/NE>
    006     D   0000FF00                        0010:W      EQ
    005     E   0000AF00                          20:B      NE
    004     E   00000100                    00000100:L      EQ
    003     E   00000020                1.235678e-12:S      EQ
    002     E   00000010   1.2345678901234567e-123:D      EQ
:
```

4.8 BREAKPOINT

BREAKPOINT	Sets breakpoints based on the address of instruction execution
BP	

Format BREAKPOINTΔ<instruction address>[Δ<count>](RET)

Parameters	
• <instruction address>	Specifies the address of the breakpoint.
• <count>	Specifies the count of fetchings of instructions at the specified address (H'1 to H'3FFF). The count is automatically set at 1 unless specified.

Function Specifies the breakpoints based on the address of instruction execution.
Program execution stops at the break address when the break conditions are satisfied.
The instruction at the break address is not executed.
Up to 255 breakpoints can be set.
Note that breakpoints are automatically enabled when a breakpoint is set.

Notes (1) If a breakpoint is set at any address other than the first byte of an instruction, the break will not be detected.
(2) The count of passes is reset when program execution stops.

Example To set a breakpoint so that execution stops when attempting to execute the instruction at address H'2000 for the eighth time:

```
: BREAKPOINT 2000 8 (RET)
:
```

4.9 BREAKPOINT_DISPLAY

BREAKPOINT_DISPLAY	Displays breakpoints based on the address of instruction execution
BPD	

Format BREAKPOINT_DISPLAY(RET)

Function Displays the breakpoints based on the address of instruction execution in the following format:

<INDEX>:	Break No.
<E/D>:	Enable/disable
<ADDRESS>:	Breakpoint address
<COUNT>:	Count of fetchings of instructions at the specified address

Example To display the current settings (address and counts are displayed in hexadecimal):

```
: BREAKPOINT_DISPLAY(RET)
  <INDEX> <E/D> <ADDRESS> <COUNT>
    000     E   00002000      8
:
```

4.10 BREAKREGISTER

BREAKREGISTER	Sets break conditions based on the value of data in a register
BR	

Format BREAKREGISTERΔ<register>[Δ<data>[;<size>][Δ<option>]](RET)

Parameters

- <register>
  Specifies the register for which the break is to be set.
- <data>
  Specifies the data value for the break condition.
  When no value is specified, a break occurs when the specified register is written.
- <size> Data size {B|W|L|S|D}
  When no size is specified, the register size is used.
  When a floating-point value is specified as the data, the size must not be omitted.

B (byte):	Byte data
W (word):	Word data
L (long):	Long-word data
S (single Float):	Single-precision floating-point data
D (double Float):	Double-precision floating-point data (only for SH-4)

- <options> Data equal/not equal {EQ|NE}

EQ (equal):	Breaks when the data are equal. (default)
NE (not equal):	Breaks when the data are not equal.

Function This command sets break conditions based on data written to the registers.
SP can be specified instead of R15.
The command sets a break condition so that execution stops when the specified register is accessed. Note that breakpoints are automatically enabled when a breakpoint is set.
Up to eight breakpoints can be set.

Examples (1) To set a breakpoint so that execution stops when register R0 is written:

```
: BREAKREGISTER R0 (RET)
:
```

(2) To set a breakpoint so that execution stops when the contents of register R1 becomes FF:

```
: BREAKREGISTER R1 FF;B (RET)
:
```

(3) To set a breakpoint so that execution stops when register R2 is written by a value other than FF:

```
: BREAKREGISTER R2 FF;B NE (RET)
:
```

(4) To set a breakpoint so that execution stops when the contents of register FR1 becomes 1.0E-5:

```
: BREAKREGISTER FR1 1.0E-5;S (RET)
:
```

4.11 BREAKREGISTER_DISPLAY

BREAKREGISTER_DISPLAY	Displays break conditions based on the value of data in a register
BRD	

Format BREAKREGISTER_DISPLAY(RET)

Function Displays the break conditions based on the value of data in a register in the following format:

<INDEX>:	Break No.
<E/D>:	Enable/disable
<REGISTER>:	Register name
<DATA>:	Written data and data size
<EQ/NE>:	Data equal/not equal

Example To display the currently set breakpoints (displayed in the floating-point format when the data size is S or in hexadecimal for other data):

```
: BREAKRESISTER_DISPLAY(RET)
  <INDEX> <E/D> <REGISTER>  <DATA>         <EQ/NE>
    003     E     FR1         1.000000e-5    EQ
    002     E     R2             000000FF    NE
    001     E     R1             000000FF    EQ
    000     E     R0       --------          EQ
```

4.12 BREAKSEQUENCE

BREAKSEQUENCE	Sets breakpoints with execution sequence specified
BS	

Format BREAKSEQUENCEΔ<instruction address1>[Δ<instruction address2>...
Δ<instruction address8>] (RET)

Parameter • <instruction address> Specifies the address(es) for sequential breakpoint

Function This command sets breakpoints with execution sequence specified.
Execution stops at the last specified address when the instructions at the specified addresses have been executed in the specified order.
Note that a sequence of up to eight addresses can be specified.

Notes (1) If a breakpoint is set at any address other than the first byte of an instruction, the break will not be detected.
(2) The execution sequence status is reset when instruction execution stops.

Example To set sequential breakpoints at addresses H'2000, H'2100 and H'3000:

```
: BREAKSEQUENCE 2000 2100 3000 (RET)
:
```

Break will occur when addresses H'2000, H'2100, and H'3000 are executed.
Note that passing an address is defined as passing at least once. Thus, the breakpoint sequence is not reset when an address is executed more than once.

4.13 BREAKSEQUENCE_DISPLAY

BREAKSEQUENCE_DISPLAY	Displays breakpoints with execution sequence specified
BSD	

Format BREAKSEQUENCE_DISPLAY(RET)

Function Displays the following information on the breakpoints with execution sequence specified:

<INDEX>:	Break No.
<E/D>:	Enable/disable
1ST BREAK POINT = xxxxxxxx:	Instruction address 1
2ND BREAK POINT = xxxxxxxx:	Instruction address 2
:	:

Example To display the currently set sequential breakpoint:

```
: BREAKSEQUENCE_DISPLAY(RET)
  <INDEX> <E/D>
    008     E  1ST BREAK POINT = 00002000
               2ND BREAK POINT = 00002100
               3RD BREAK POINT = 00002200
:
```

4.14 COMPARE

COMPARE	Compares memory contents
CMP	

Format COMPAREΔ<start address>Δ<end address>
Δ<comparison memory start address>(RET)

Parameters
- <Start address>
  Specifies the start address of the source data.
- <End address>
  Specifies the end address of the source data.
- <Comparison memory start address>
  Specifies the start of the comparison data memory area.

Function Compares the specified range of memory (the source data) with the comparison data in byte units.
When data that does not match is found, those data items and their addresses are displayed.

Example To compare the H'500 bytes of data starting at address H'1000 with the H'500 bytes of data starting at address H'2000, and to display the addresses and values of the source data and comparison data when data which does not match is found:

```
: COMPARE 1000 14FF 2000(RET)
SOURCE DATA          COMPARED DATA
00001005 3F          00002005 42
     :                    :
000014FE 00          000024FE 80
:
```

4.15 DATA_SEARCH

DATA_SEARCH	Searches for data
DS	

Format DATA_SEARCHΔ<start address>Δ<end address>Δ<data>[;<size>](RET)

Parameters
- <start address>
  Specifies the search start address.
- <end address>
  Specifies the search end address.
- <data>
  Specifies the data to be searched.
- <size> Size of data {B|W|L|D|S}

B (byte):	Searches for byte data (default).
W (word):	Searches for word data.
L (long):	Searches for long-word data.
D (double float):	Double-precision floating-point data
S (single float):	Single-precision floating-point data

Function This command searches for the specified data in the specified memory range.

Note When searching for word data, the start address must be the word boundary (multiple of two). When searching for single-precision floating-point, double-precision floating-point, or long-word data, the start address must be the long-word boundary (multiple of four).

Example To search for the value 005E from address H'1000 to address H'14FF:

```
: DATA_SEARCH 1000 14FF 005E;W (RET)
ADDRESS
00001004
00001100
000011A8
:
```

4.16 DISASSEMBLE

DISASSEMBLE	Disassembles and displays memory contents
DA	

Format DISASSEMBLEΔ<start address>[Δ<instruction count>](RET)

Parameters
- <start address>
  Specifies the start address for disassembly.
- <instruction count>
  Specifies the count of instructions to be disassembled (default = 16, max. = 65535).

Function Disassembles and displays the range specified by the start address the instruction count.
Displays the instruction start address, mnemonic instruction and operand.
An invalid instruction is displayed with a hexadecimal instruction code.

Example To disassemble and display the four instructions starting at address H'400:

```
: DISASSEMBLE 400 4 (RET)
00000400  STS.L   PR,@-R15
00000402  ADD     #H'C8,R15
00000404  MOV     #H'00,R3
00000406  MOV.L   R3,@(H'08:4, R15)
:
```

4.17 DISPLAY_CHARACTERS

DISPLAY_CHARACTERS	Displays character string
DCH	

Format DISPLAY_CHARACTERSΔ<character string>(RET)

Parameter • <character string> Specifies character string.

Function Displays the characters following a space behind the command name.

Example To display SIMULATOR on the screen:

```
: DISPLAY_CHARACTERS SIMULATOR (RET)
SIMULATOR
:
```

4.18 EXEC_MODE

EXEC_MODE	Switches execution mode
EM	

Format Set: EXEC_MODEΔ{S|C}(RET)
Display: EXEC_MODE(RET)

Parameter • Execution mode specifier {S|C}

S (stop):	In this mode, execution is stopped when the simulator/debugger detects an abnormality (simulation error) in the user program.
C (continue):	In this mode, simulation errors are ignored and execution continues when the simulator/debugger detects an abnormality (simulation error) in the user program.

The simulator/debugger execution mode is set to S when first invoked.

Function This command selects whether execution will continue or stop when an abnormality is detected during the execution of user program. When the execution mode specifier is omitted, the current setting of the execution mode is displayed.
Refer to section 2.12 (2), Break due to detection of an error during execution of the user program, for more information on abnormalities which may occur while executing the user program.

Description

Set:	Stop mode is recommended for the early stages of debugging, with continue mode being useful in the later stages.
Display:	Stop is displayed in stop mode, and Continue in continue mode.

Examples (1) To set the execution mode to continue mode:

```
: EXEC_MODE C (RET)
:
```

(2) To display the current execution mode:

```
: EXEC_MODE (RET)
Continue
:
```

4.19 FILE_LOAD

FILE_LOAD	Loads file
FL	

Format FILE_LOADΔ<file name>[Δ{SYS|ELF|STY}](RET)

Parameters

- <file name>
  Specifies the name of the file to be loaded.
  An extension is added as follows if the file format is specified, or ".abs" is added as a file extension if the format specified:
  SYS: .abs
  ELF: .abs
  STY: .mot
- File format specifications {SYS|ELF|STY}

SYS (SYSROF):	Loads a SYSROF file (default).
ELF (ELF):	Loads an ELF file.
STY (STYPE):	Loads an STYPE file (Motorola S record type only).

Function Loads the user program.
Although memory required for loading is allocated by the FILE_LOAD command for SYSROF and ELF files, no memory is allocated for STYPE files.

Description Reset the simulator/debugger before loading the user program (for SYSROF and ELF only).
The default settings after loading the user program are as follows:

Memory area: An area for the user program is allocated (for SYSROF and ELF only).

PC: SYSROF:
If an entry address was specified in the user program, the PC is set to that address. Otherwise, the PC is set to the start address of the code section that appeared first.
STYPE:
If an entry address was specified in the user program, the PC is set to that address. Otherwise, the PC is set to the start address of the load module.
ELF:
The PC is set to the start address of the section that appeared first.

SP: The SP is set to the last address of the internal RAM + 1.
If no internal RAM area exists, the SP is set to 0.

Other registers and flags are not set.

Examples (1) To load SYSROF-type user program "test1.abs":

```
: FILE_LOAD test1.abs (RET)
:
```

(2) To load STYPE file "test2.mot":

```
: FILE_LOAD test2.mot STY (RET)
:
```

4.20 FILE_SAVE

FILE_SAVE	Saves memory data to a file
FS	

Format FILE_SAVEΔ<file name>Δ<start address>Δ<end address>(RET)

Parameters
- <file name>
  Specifies the file name to be saved.
- <start address>
  Specifies the start address of memory data to be saved.
- <end address>
  Specifies the end address of memory data to be saved.

Function Stores the memory area to a file. Data is saved in the Motorola S record format.
When a file name which already exists is specified, data is overwritten.
When no extension is specified for a file name, ".mot" is added.

Example To save memory data from address H'2000 to address H'2FFF in file "sample.mot":

```
: FILE_SAVE sample.mot 2000 2FFF (RET)
:
```

4.21 GO

GO	Executes instructions continuously
G	

Format GOΔ[<start address>][;D](RET)

Parameters
- <start address>
  Specifies the address from which program execution starts.
  When omitted, execution starts from the address specified by the program counter.
- Break disable
  D (disable breaks): Breakpoints specified with the break commands are temporarily disabled.

Function This command executes the user program continuously starting at the specified start address.
When D is specified, the break is temporarily disabled during GO command execution but are enabled again when execution stops.
When execution stops, the executed instruction count (in decimal), the current register values, a disassembled display of the last instruction executed, and a termination message are displayed.
After specifying a start address, the pipeline is reset at execution start.

Example To execute the user program from address H'1000 to address H'101E (for SH-1):

```
: GO 1000(RET)
Exec Instructions = 30
PC=00001020 SR=00000000:----------------------IIII---- SP=05000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 0000FFFF 00000000 00000000 01000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000010 00000000 0000FFFF 00000000 00000000 05000000
0000101E MOV       #H'00,R3
+++5001 : PC breakpoint
:
```

4.22 GO_RANGE

GO_RANGE	Executes instructions continuously (with range specified)
GR	

Format GO_RANGEΔ<start address>Δ<break address>[;D](RET)

Parameters
- <start address>
  Specifies the address from which program execution starts.
- <break address>
  Specifies the address at which program execution stops.
- Break disable
  D (disable breaks): Breakpoints specified with the break commands are temporarily disabled.

Function This command executes the user program continuously starting at the specified start address and stops execution at the specified break address. The instruction at the break address is not executed.
When D is specified, the break is temporarily disabled during GO command execution but are enabled again when execution stops.
When execution stops, the executed instruction count (in decimal), the current register values, a disassembled display of the last instruction executed, and a termination message are displayed.
After specifying a start address, the pipeline is reset at execution start.

Note If a break address is specified at a point that is not the first byte of an instruction, the break will not be detected.

Example To execute the user program from address H'1000 to address H'1020 for SH-3(the instruction at address H'1020 is not executed):

```
: GO_RANGE 1000 1020(RET)
Exec Instructions = 30
PC=00001020 SR=700000F0:-MRB--------------------1111 SP=00001FEC
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000003 00000002 000001E4 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000001 000001EC 00001FEC
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000
PTEH=00000000 PTEL=00000000 TTB=00000000 TEA=00000000 MMUCR=00000000
EXPEVT=00000000 INTEVT=00000000 TRA=00000000 CCR=00000000
0000101E MOV       #H'00,R3
+++5001 : PC breakpoint
:
```

4.23 GO_RESET

GO_RESET	Executes user program from the vector address
GS	

Format GO_RESET(RET)

Function Executes the user program starting from the address specified by the reset vector. When execution stops, the executed instruction count (in decimal), the current register values, a disassembled display of the last instruction executed, and a termination message are displayed.

Description SH-1/SH-2/SH-DSP/SH-2E series:
Before execution, the reset exception processing vector table must be set on memory. Save the initial values of PC and SP on the table.

Table 4.2 Vector Table

Register	Vector No.	Vector Table Address to be Saved
PC	0	H'00000000 to H'00000003
SP	1	H'00000004 to H'00000007

SH-3/SH-3E/SH-4 series:
The reset vector address is fixed at H'A0000000.

Example To start power-on reset execution from the reset vector (for SH-3):

```
: GO_RESET

Exec Instructions = 12

PC=A0000018 SR=700000F0:-MRB--------------------1111---- SP=7F001000

GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000

R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 7F001000

R0_BANK-R3_BANK 00000000 00000000 00000000 00000000

R4_BANK-R7_BANK 00000000 00000000 00000000 00000000

SSR=00000000 SPC=00000000

PTEH=00000000 PTEL=00000000 TTB=00000000 TEA=00000000 MMUCR=00000000
EXPEVT=00000000 INTEVT=00000000 TRA=00000000 CCR=00000000

A0000016    NOP

+++5001 : PC breakpoint

:
```

4.24 GO_TILL

GO_TILL	Executes instructions continuously (with stop address specified)
GT	

Format GO_TILLΔ<break address 1>[Δ<break address 2>Δ<break address 3>... Δ<break address 10>][;D](RET)

Parameters
- <break address>
  Specifies the address at which user program execution stops. (up to 10 points)
- Break disable
  D (disable breaks): Breakpoints specified with the break commands are temporarily disabled.

Function This command executes the user program continuously starting at the address specified by the program counter, and stops execution at specified break address. The instruction at the break address is not executed.
Up to ten break addresses can be specified.
When D is specified, the break is temporarily disabled during GO command execution but are enabled again when execution stops.
When execution stops, the executed instruction count (in decimal), the current register values, a disassembled display of the last instruction executed, and a termination message are displayed.

Note If a break address is specified at a point that is not the first byte of an instruction, the break will not be detected.

Example To continuously execute the user program from the address specified by the current program counter to address H'1000, H'1010 or H'1020 for SH-1:

```
: GO_TILL 1000 1010 1020(RET)
Exec Instructions = 30
PC=00001020 SR=00000000:-------------------------------- SP=05000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 0000FFFF 00000000 00000000 01000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000010 00000000 0000FFFF 00000000 00000000 05000000
0000101E  MOV      #H'00,R3
+++5001 : PC breakpoint
:
```

4.25 HELP

HELP	Displays command name and input format
HE	

Format HELP[Δ<command name>](RET)

Parameter • <command name>
Specifies the name of the command to display the help message.

Function Displays the help message for the specified command.
When the command name is omitted, a list of commands is displayed.

Examples (1) To display a list of commands:

```
: HELP(RET)
.<register>                 ;
ASsemble                     Break_Clear
Break_Enable                 BreakAccess
BreakAccess_Display          BreakData
BreakData_Display            BreakPoint
BreakPoint_Display           BreakRegister
       :                            :
Trace                        Trace_cLear
Trace_Condition              Trap_Address
Trap_address_Display         Trap_address_Enable
:
```

(2) To display the syntax of the HELP command:

```
:HELP HELP(RET)
HE|HELP[<command name>]
:
```

4.26 LOAD_STATUS

LOAD_STATUS	Restores the simulator/debugger memory and register status
LS	

Format LOAD_STATUS[Δ<file name>](RET)

Parameter • <file name>
Specifies the name of a file in which the simulator/debugger memory and register status is saved.
When the file name is omitted, the file sdsh.sav is assumed.
When the file extension is omitted, the extension .sav is supplied as default.

Function The states of memory and the registers are restored to the point when the corresponding SAVE_STATUS command was executed.
Reloads the load module which has been loaded at SAVE_STATUS command execution.

Example To load the memory and register status saved in the file "test1.sav":

```
: LOAD_STATUS test1.sav(RET)
:
```

4.27 LOG

LOG	Starts creating an execution history file
LO	

Format LOGΔ<file name>[ΔA](RET)

Parameters
- <file name>
  Specifies the file name to which the execution history is output.
- Append mode specification
  A (append): Adds the execution history to the specified file.
  When this option is omitted, the execution history is saved from the start of the specified file.

Function Starts outputting to the execution history file.
If the specified file already exists, the file is deleted to create a new file.
When restart is specified without stopping after start, the file which is receiving outputs is closed and then output to the specified file resumes.

Note When an error occurs during I/O processing of a system call, I/O data is not written to the output file.

Examples (1) To start writing to sample.log file to which command inputs and display data are written:

```
: LOG sample.log (RET)
:
```

(2) To add the execution history to sample.log:

```
: LOG sample.log A (RET)
:
```

4.28 LOG_ENABLE

LOG_ENABLE	Enables/disables execution history file creation
LE	

Format LOG_ENABLEΔ{E|D}(RET)

Parameter • Terminates and resumes outputting to the execution history file {E|D}
E (enable): Resumes outputting to the file.
D (disable): Terminates outputting to the file.

Function Outputting to the file terminates when option "D" (disable) is specified or resumes when "E" (enable) is specified.

Note When an error occurs during I/O processing of a system call, I/O data is not written to the output file.

Examples (1) To terminate writing to the file:

```
: LOG_ENABLE D (RET)
:
```

(2) To resume writing to the file:

```
: LOG_ENABLE E (RET)
:
```

4.29 LOG_STOP

LOG_STOP	Stops creating an execution history file
LT	

Format LOG_STOP(RET)

Function Stops creating an execution history file.

Note When an error occurs during I/O processing of a system call, I/O data is not written to the output file.

Example To stop writing to the file:

```
:  LOG_STOP (RET)
:
```

4.30 MAP_CLEAR

MAP_CLEAR	Clears memory areas
MC	

Format MAP_CLEARΔ<start address>Δ<end address>(RET)

Parameters
- <start address>
  Specifies the start address in the memory area.
- <end address>
  Specifies the end address in the memory area.

Function Clears memory areas allocated by the MAP_SET command.

Example To clear address H'301F from address H'3000 which has already been specified:

```
: MAP_CLEAR 3000 301F (RET)
:
```

4.31 MAP_DISPLAY

MAP_DISPLAY	Displays memory areas
MI	

Format MAP_DISPLAY[ΔM](RET)

Parameter • Memory map information display

M (map): Specifies the memory map information display in the CPU information file.

Functions The memory map information is displayed in the following format.

(1) Memory map information display

<START>:	Start address
<END>:	End address
<ATTR>:	Access type (R: read, W: write, RW: read/write)
<SECT_NAME>:	Section name

(2) Memory map information display in the CPU information file

<KIND>:	Memory type (I/O: internal I/O, RAM: internal RAM, ROM: internal ROM, EXT: external bus area)
<START>:	Start address
<END>:	End address
<STATE>:	Memory access state count (--- is displayed for SH-4)
<BUS>:	Memory data bus width

Examples (1) To display the current memory allocation state:

```
: MAP_DISPLAY (RET)
  <START>  <END>      <ATTR>  <SECT_NAME>
  00000000-000003FF      W
  00002000-000020EF     RW    SECT1
  00003000-0000301F      W
:
```

(2) To display memory map information in the CPU information file:

```
: MAP_DISPLAY M (RET)
  <KIND> <START>  <END>      <STATE>     <BUS>
   EXT   00000000-7EFFFFFF   00000001   00000032
   RAM   7F000000-7F000FFF   00000001   00000032
   EXT   7F001000-DFFFFFFF   00000001   00000032
   I/O   E0000000-FFFFFFFF   00000001   00000032
:
```

4.32 MAP_SET

MAP_SET	Sets memory areas
MS	

Format MAP_SETΔ<start address>[Δ<end address>][Δ{R|W|RW}](RET)

Parameters
- <start address>
  Specifies the start address in the memory area.
- <end address>
  Specifies the end address in the memory area.
  The start address is assumed unless specified.
- Access type {R|W|RW}

R (read):	Specifies the memory area to be read-only.
W (write):	Specifies the memory area to be write-only.
RW (read/write):	Specifies the memory area to be read/write. (default)

Function Sets memory areas used by the user program.

Notes (1) Areas are reset with newly specified contents even if the range to be set by the MAP_SET command has already been allocated.
(2) Several areas can be re-specified or cleared at the same time.

Examples (1) To allocate addresses H'3000 to H'301F as a read-only memory area:

```
: MAP_SET 3000 301F R(RET)
:
```

(2) To change the access type to write-only for the memory area allocated to addresses H'0 to H'03FF :

```
: MAP_SET 0 3FF W(RET)
:
```

4.33 MEMORY_DISPLAY

MEMORY_DISPLAY	Displays memory contents
MD	

Format MEMORY_DISPLAYΔ<start address>[Δ<length>][;<size>](RET)

Parameters
- <start address>
  Specifies the start address of memory content display.
- <length>
  Specifies the length (byte count) of the data to be displayed
  (default: H'100, max.: H'4000)
- <size> Data size {B|W|L|D|S|A}

B (byte):	Byte data (default)
W (word):	Word data
L (long):	Long-word data
D (double float):	Double-precision floating-point data
S (single float):	Single-precision floating-point data
A (ASCII):	ASCII data

Function Displays memory contents.

Note When memory contents are displayed in a word size, the start address must be the word boundary (multiple of two). When memory contents are displayed in a single-precision floating-point, double-precision floating-point, or long-word size, the start address must be the long-word boundary (multiple of four).

Examples (1) To display memory contents from address H'1000 in byte units:

```
: MEMORY_DISPLAY 1000;B(RET)
address  +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00001000 4F 22 7F C8 E3 00 1F 32 A0 12 00 09 D1 1E 41 0B
00001010 00 09 1F 03 40 11 89 01 60 0B 1F 03 53 F2 43 08
    :                           :
000010E0 A0 08 00 09 63 F2 52 F2 32 38 1F 23 E3 0A051 FS
000010F0 31 33 89 D1 63 F2 52 F3 32 3C 1F 23 E3 0A 51 F3
:
```

(2) To display 16-byte memory contents from address H'1000 in word units:

```
: MEMORY_DISPLAY 1000 10;W(RET)
address  +0   +2   +4   +6   +8   +A   +C   +E
00001000 4F22 7FC8 E300 1F32 A012 0009 D11E 410B
:
```

(3) To display 16-byte memory contents from address H'1000 in long word units:

```
: MEMORY_DISPLAY 1000 10;L(RET)
address  +0       +4       +8       +C
00001000 4F227FC8 E3001F32 A0120009 D11E410B
:
```

(4) To display 8-byte double-precision floating-point data from address H'2000:

```
: MEMORY_DISPLAY 2000 8;D(RET)
address  +0 +1 +2 +3 +4 +5 +6 +7
00002000 23 1F 00 E3 C8 7F 22 4F 2.87495706857453e-67
:
```

(5) To display 8-byte single-precision floating-point data from address H'2000:

```
: MEMORY_DISPLAY 2000 8;S(RET)
address  +0 +1 +2 +3


00002000 C8 7F 22 4F -2.612572e+05
00002004 32 1F 00 E3 9.255220e-09
:
```

(6) To display 22-byte ASCII data from address H'3000:

```
: MEMORY_DISPLAY 3000 16;A(RET)
address       ASCII
00003000 0".....2......A.
00003010 ....@.
:
```

4.34 MEMORY_EDIT

MEMORY_EDIT	Modifies memory contents
ME	

Format Modify: MEMORY_EDITΔ<start address>Δ<data>[;<size>](RET)
Interactive mode: MEMORY_EDITΔ<start address>[;<size>](RET)

Parameters
- <start address>
  Specifies the start address to be modified.
  <data>
  Specifies the contents to be modified.
- <size> Data size {B|W|L|D|S|A}

B (byte):	Memory is to be modified in byte units. (default)
W (word):	Memory is to be modified in word units.
L (long):	Memory is to be modified in long-word units.
D (double float):	Memory is to be modified in double-precision floating-point units.
S (single float):	Memory is to be modified in single-precision floating-point units.
A (ASCII):	Memory is to be modified in ASCII character string units.

Function Changes the contents of memory to the specified value.

Description When a command for specifying the interactive mode is input, the interactive mode is entered after the contents of the specified address is displayed.

```
MEMORY_EDITΔ<start address>(RET)
address data: [{<data>|^}](RET)
address data: [{<data>|^}](RET)
      .
      .
address data: .(RET)
```

address data:	Displays the data before modification.
<data>:	Specifies the data to be modified.
^:	Displays the contents of the previous address.
Only (RET) is input:	Displays the contents of the next address.
. (period):	Terminates the MEMORY_EDIT command.

Note When memory contents are modified in a word size, the start address must be the word boundary (multiple of two). When memory contents are modified in a single-precision floating-point size, double-precision floating-point size, or long-word size, the start address must be the long-word boundary (multiple of four).

Examples (1) To change the contents of one byte of memory at address H'1000 to H'3E:

```
: MEMORY_EDIT 1000 3E;B(RET)
:
```

(2) To change the memory contents in one byte unit from address H'1000 in the interactive form:

```
: MEMORY_EDIT 1000;B(RET)
00001000 3E : 5F(RET)
00001001 FF : (RET)
00001002 55 : 25(RET)
   :        :
00001005 CC : .(RET)
:
```

(3) To change the memory contents in single-precision floating-point units from address H'2000 in the interactive form:

```
: MEMORY_EDIT 2000;S(RET)
00002000 1.413991E-3 : F'-3.1415922E+1(RET)
00002004 1.234567E+5 : .(RET)
:
```

4.35 MEMORY_FILL

MEMORY_FILL	Initializes memory areas
MF	

Format MEMORY_FILLΔ<start address>Δ<end address>Δ<data value>[;<size>]
[Δ<verify flag>](RET)

Parameters
- <start address>
  Specifies the start address of the memory to be initialized.
- <end address>
  Specifies the end address of the memory to be initialized.
- <data value>
  Specifies the data to be set.
- <size> Data size {B|W|L|D|S}

B (byte):	Byte data (default)
W (word):	Word data
L (long):	Long-word data
D (double float):	Double-precision floating-point data
S (single float):	Single-precision floating-point data

- <verify flag> Data is verified after setting {V|N}
  V: Verified (default)
  N: Not verified

Function Sets initial data in the specified range of addresses.

Note When word-size data is written into a memory area range, the start address must be the word boundary (multiple of two). When single-precision floating-point size, double-precision floating-point size, or long-word size is written into a memory area range, the start address must be the long-word boundary (multiple of four).

Examples (1) To clear addresses H'1000 to H'1FFF to 0 and then verify them:

```
: MEMORY_FILL 1000 1FFF 0(RET)
:
```

(2) To set H'FF00 to addresses H'2000 to H'2FFF in word units without verifying them:

```
: MEMORY_FILL 2000 2FFF FF00;W N(RET)
:
```

4.36 MEMORY_MOVE

MEMORY_MOVE	Moves memory blocks
MV	

Format MEMORY_MOVEΔ<start address>Δ<end address>
Δ<transfer destination address>(RET)

Parameters

• <start address>:	Specifies the start address of the transfer source.
• <end address>:	Specifies the end address of the transfer source.
• <transfer destination address>:	Specifies the start address of the transfer destination.

Function Copies the memory data in the specified range to the specified transfer destination. Before copying, allocate an area for the transfer destination using the MAP_SET command.

Example To copy the contents in addresses H'1000 to H'14FF one by one to addresses H'2000 and later:

```
: MEMORY_MOVE 1000 14FF 2000 (RET)
:
```

4.37 PERFORMANCE_ANALYSIS

PERFORMANCE_ANALYSIS	Sets execution performance analysis
PA	

Format PERFORMANCE_ANALYSIS[Δ<start address>](RET)

Parameter • <start address>
Specifies the start address of the function whose execution performance is to be analyzed.
When no start address is specified, execution performance of all functions (only those actually executed) is to be analyzed.

Function Analyzes the maximum, minimum and total execution cycle and call counts of the specified function(s).

Note The analysis results may be invalid if the PC value is changed (pipeline reset) by the .<register> or other commands during function analysis.

Example To specify execution performance analysis of all functions:

```
:PERFORMANCE_ANALYSIS(RET)
:
```

4.38 PERFORMANCE_ANALYSIS_CLEAR

PERFORMANCE_ANALYSIS_CLEAR	Clears execution performance analysis
PC	

Format PERFORMANCE_ANALYSIS_CLEAR[Δ<index>](RET)

Parameter • <index>
Specifies the function number to be cleared (index can be checked by execution performance analysis display).
If no index is specified, all analysis results are cleared.

Function Clears execution performance analysis.

Example To clear the analysis results of index no.1:

```
:PERFORMANCE_ANALYSIS_CLEAR 1(RET)
:
```

4.39 PERFORMANCE_ANALYSIS_DISPLAY

PERFORMANCE_ANALYSIS_DISPLAY	Displays execution performance analysis results
PD	

Format PERFORMANCE_ANALYSIS_DISPLAY[Δ{A|C}](RET)

Parameter • Display type {A|C}

A (address):	Displays analysis results in the address (ascending) order (default).
C (cycle):	Displays analysis results in the cycle count (descending) order.

Function Displays the execution performance analysis results in the following format:

INDEX:	Registered number of the function
ADDRESS:	Start address of the function
MAXCYCLE:	Maximum execution cycle count of the function
MINCYCLE:	Minimum execution cycle count of the function
TOTALCYCLE:	Total execution cycle count of the function
COUNT:	Call count of the function
%:	Ratio of the total execution cycle count of the function to that of the whole user program
HISTOGRAM:	Displays the above ratio in a histogram

Description (1) Displays the execution cycle count for each specified function.

(2) The execution cycle count is obtained from the difference between the accumulative execution cycle counts at execution of call instructions from the specified function and that of return instructions from the specified function.

(3) Up to 9999 functions can be set.

Examples (1) To display the analysis results in the address (ascending) order:

```
:PERFORMANCE_ANALYSIS_DISPLAY(RET)
INDEX ADDRESS  MAXCYCLE   MINCYCLE   TOTALCYCLE COUNT    % HISTOGRAM
  0   00001234      20000      10000      50000       3 45 ####
  1   00005678      15000       5000     600000       9 55 #####
:
```

(2) To display the analysis results in the cycle count (descending) order:

```
:PERFORMANCE_ANALYSIS_DISPLAY C(RET)
INDEX ADDRESS  MAXCYCLE   MINCYCLE   TOTALCYCLE COUNT    % HISTOGRAM
  1   00005678      15000       5000     600000       9 55 #####
  0   00001234      20000      10000      50000       3 45 ####
:
```

4.40 PERFORMANCE_ANALYSIS_ENABLE

PERFORMANCE_ANALYSIS_ENABLE	Enables/disables or resets execution performance analysis
PE	

Format PERFORMANCE_ANALYSIS_ENABLEΔ{E|D|R}(RET)

Parameter • Specification type {E|D|R}

E (enable):	Enables analysis.
D (disable):	Disables analysis.
R (reset):	Resets the analysis results.

Function Enables/disables execution performance analysis or resets the analysis results.

Description (1) Execution performance analysis is disabled when the simulator is initiated.

(2) By resetting, only the analysis results are reset and the execution performance analysis enable/disable setting and the registered start address (including all functions) are not changed.

Examples (1) To disable execution performance analysis:

```
: PERFORMANCE_ANALYSIS_ENABLE D(RET)
:
```

(2) To reset the analysis results:

```
: PERFORMANCE_ANALYSIS_ENABLE R(RET)
:
```

4.41 QUIT

QUIT	Exits the simulator/debugger
Q	

Format QUIT(RET)

Function Exits the simulator/debugger and returns to the OS.
Closes the execution history and command files if they are opened.

Example To terminate simulator/debugger processing:

```
: QUIT (RET)
```

(The csdsh and the simulator/debugger is terminated and the OS prompt will be displayed)

4.42 RADIX

RADIX	Sets the radix
RX	

Format Set: RADIXΔ{B|O|D|H}(RET)
Display: RADIX(RET)

Parameter • Radix {B|O|D|H}
B: Sets the radix to binary.
O: Sets the radix to octal.
D: Sets the radix to decimal.
H: Sets the radix to hexadecimal.
The radix is set to hexadecimal when the simulator/debugger is invoked.

Function Sets or displays the default radix. The radix is displayed if no parameter is specified. When B', H', D' or O' is input before numeric data, the input precedes the default radix.
Display contents
B: Binary
O: Octal
D: Decimal
H: Hexadecimal

Examples (1) To display the current radix:

```
: RADIX(RET)
Hexadecimal
:
```

(2) To change the radix to decimal:

```
: RADIX D(RET)
: RADIX(RET)
Decimal
:
```

4.43 REGISTER

REGISTER	Displays registers
R	

Format REGISTER[Δ{C|F|A}](RET)

Parameter Specification of the display register {C|F|A}

C (cpu):	Displays the CPU register contents (and DSP registers for the SH-DSP) (default)
F (fpu):	Displays the FPU register contents (Valid for only for the SH-2E, SH-3E, and SH-4)
A (all):	Displays the contents of the CPU, FPU, and management registers (display of the management register is valid only for the SH-3, the SH-3E, and SH-4).

Function Displays the following register contents.

Display for the SH-1 or SH-2 series

• CPU registers	
—General registers:	R0-R15
—Control registers:	SR, GBR, VBR
—System registers:	MACH, MACL, PR, PC

Display for the SH-DSP series

• CPU registers	
—General registers:	R0-R15
—Control registers:	SR, RS, RE, GBR, VBR, MOD
—System registers:	MACH, MACL, PR, PC
• DSP registers	
—Data registers:	A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1
—Control register:	DSR

Display for the SH-3 series

• CPU registers	
—General registers:	R0-R15, R0_BANK-R7_BANK
—Control registers:	SR, GBR, VBR, SSR, SPC
—System registers:	MACH, MACL, PR, PC
• Management registers:	PTEH, PTEL, TTB, TEA, MMUCR, EXPEVT, INTEVT, TRA, CCR

Display for the SH-2E series

- CPU registers

—General registers:	R0-R15
—Control registers:	SR, GBR, VBR
—System registers:	MACH, MACL, PR, PC

- FPU registers:

—Floating-point registers:	FR0-FR15
—Control registers:	FPSCR
—System registers:	FPUL

Display for the SH-3E series:

- CPU registers

—General registers:	R0-R15 R0_BANK-R7_BANK
—Control registers:	SR, GBR, VBR, SSR, SPC
—System registers:	MACH, MACL, PR, PC

- Management registers: PTEH, PTEL, TTB, TEA, MMUCR, EXPEVT, INTEVT, TRA, CCR
- FPU registers

—Floating point registers:	FR0-FR15
—Control register:	FPSCR
—System register:	FPUL

Display for the SH-4 series:

- CPU registers

—General registers:	R0-R15 R0_BANK-R7_BANK
—Control registers:	SR, GBR, VBR, SSR, SPC, SGR, DBR
—System registers:	MACH, MACL, PR, PC

- Management registers: PTEH, PTEL, TTB, TEA, MMUCR, EXPEVT, INTEVT, TRA, CCR, QACR0, QACR1
- FPU registers

—Floating point registers:	FR0-FR15, XF0-XF15, DR0, DR2, DR4, DR6, DR8, DR10, DR12, DR14, XD0, XD2, XD4, XD6, XD8, XD10, XD12, XD14
—Control register:	FPSCR
—System register:	FPUL

Example To display the registers:
SH-1/SH-2 series

```
: REGISTER A(RET)
PC=00000000 SR=000000F0:-----------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
:
```

SH-DSP series

```
: REGISTER A(RET)
PC=00000000 SR=000000F0:----000000000000--------111100-- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
RS=00000000 RE=00000000 MOD=00000000
DSR=00000000:----------------------------COB-
A0G=00 A0=00000000 M0=00000000 X0=00000000 Y0=00000000
A1G=00 A1=00000000 M1=00000000 X1=00000000 Y1=00000000
:
```

SH-3 series

```
: REGISTER A(RET)
PC=00000000 SR=700000F0:-MRB--------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000
PTEH=00000000 PTEL=00000000 TTB=00000000 TEA=0000000 MMUCR=00000000
EXPEVT=00000000 INTEVT=00000000 TRA=00000000 CCR=00000000
:
```

```
: REGISTER C(RET)
PC=00000000 SR=700000F0:-MRB--------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000
:
```

SH-3E series

```
: REGISTER A(RET)
PC=00000000 SR=700000F0:-MRB--------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000
PTEH=00000000 PTEL=00000000 TTB=00000000 TEA=0000000 MMUCR=00000000
EXPEVT=00000000 INTEVT=00000000 TRA=00000000 CCR=00000000
FPUL=00000000 FPSCR=00040001:-------------D----------------RZ
FR0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR0- 3  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR4- 7  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR8-11  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR12-15 0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
:
```

```
: REGISTER F(RET)
PC=00000000 SR=700000F0:-MRB------------------1111---- SP=00000000
FPUL=00000000 FPSCR=00040001:-------------D----------------RZ
FR0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR0- 3  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR4- 7  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR8-11  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR12-15 0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
:
```

```
: REGISTER C(RET)
PC=00000000 SR=700000F0:-MRB--------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000
:
```

SH-2E series

```
: REGISTER A(RET)
PC=00000000 SR=000000F0:-----------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FPUL=00000000 FPSCR=00040001:-------------D-----------------RZ
FR0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR0- 3   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR4- 7   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR8-11   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR12-15  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
:
```

```
: REGISTER F(RET)
PC=00000000 SR=000000F0:------------------------1111---- SP=00000000
FPUL=00000000 FPSCR=00040001:-------------D-----------------RZ
FR0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR0- 3   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR4- 7   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR8-11   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR12-15  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00

: REGISTER C(RET)
PC=00000000 SR=000000F0:------------------------1111---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
:
```

SH-4 series

```
: REGISTER A(RET)
PC=00000000 SR=700000F0:-P1B--------------------1111---F SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000 DBR=00000000 SGR=00000000
PTEH=00000000 PTEL=00000000 TTB=00000000 TEA=00000000 MMUCR=00000000
EXPEVT=00000000 INTEVT=00000000 TRA=00000000 CCR=00000000
QACR0=00000000 QACR1=00000000
FPUL=00000000 FPSCR=00040001:----------0SSZ----------------RZ
FR0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR0- 3   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR4- 7   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR8-11   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR12-15  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
```

```
XF0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
XF8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
XF0- 3   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF4- 7   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF8-11   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF12-15  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
DR0-6  0000000000000000 0000000000000000 0000000000000000 0000000000000000
DR8-14 0000000000000000 0000000000000000 0000000000000000 0000000000000000
DR0 ,DR2  0.000000000000000e+00 0.000000000000000e+00
DR4 ,DR6  0.000000000000000e+00 0.000000000000000e+00
DR8 ,DR10 0.000000000000000e+00 0.000000000000000e+00
DR12,DR14 0.000000000000000e+00 0.000000000000000e+00
XD0-6  0000000000000000 0000000000000000 0000000000000000 0000000000000000
XD8-14 0000000000000000 0000000000000000 0000000000000000 0000000000000000
XD0 ,XD2  0.000000000000000e+00 0.000000000000000e+00
XD4 ,XD6  0.000000000000000e+00 0.000000000000000e+00
XD8 ,XD10 0.000000000000000e+00 0.000000000000000e+00
XD12,XD14 0.000000000000000e+00 0.000000000000000e+00
:

: REGISTER F(RET)
PC=00000000 SR=700000F0:-P1B--------------------1111---F SP=00000000
FPUL=00000000 FPSCR=00040001:----------0SSZ----------------RZ
FR0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
FR0- 3   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR4- 7   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR8-11   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
FR12-15  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
XF8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
XF0- 3   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF4- 7   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF8-11   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
XF12-15  0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
```

```
DR0-6  0000000000000000 0000000000000000 0000000000000000 0000000000000000
DR8-14 0000000000000000 0000000000000000 0000000000000000 0000000000000000
DR0 ,DR2  0.000000000000000e+00 0.000000000000000e+00
DR4 ,DR6  0.000000000000000e+00 0.000000000000000e+00
DR8 ,DR10 0.000000000000000e+00 0.000000000000000e+00
DR12,DR14 0.000000000000000e+00 0.000000000000000e+00
XD0-6  0000000000000000 0000000000000000 0000000000000000 0000000000000000
XD8-14 0000000000000000 0000000000000000 0000000000000000 0000000000000000
XD0 ,XD2  0.000000000000000e+00 0.000000000000000e+00
XD4 ,XD6  0.000000000000000e+00 0.000000000000000e+00
XD8 ,XD10 0.000000000000000e+00 0.000000000000000e+00
XD12,XD14 0.000000000000000e+00 0.000000000000000e+00
:

: REGISTER C(RET)
PC=00000000 SR=700000F0:-P1B--------------------1111---F SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3_BANK 00000000 00000000 00000000 00000000
R4_BANK-R7_BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000 DBR=00000000 SGR=00000000
:
```

4.44 RESET

RESET	Resets the simulator/debugger
RS	

Format RESET(RET)

Function Resets the simulator/debugger.

After this command is executed, the simulator/debugger is set as follows:

Pipeline:	Reset
Registers:	Initialized as follows:
	• SH-1/SH-2/SH-DSP series SR: H'F0 Others: H'0
	• SH-2E series SR: H'F0 FPSCR: H'40001 Others: H'0
	• SH-3 SR: H'700000F0 Others: H'0
	• SH-3E/SH-4 SR: H'700000F0 FPSCR: H'40001 Others: H'0
Memory:	All memory settings are cleared.
User program:	All information concerning the user program is deleted, and the simulator/debugger assumes that no program is loaded.
Command:	All command settings excluding the RADIX command are cleared and initialized.

Example To reset the simulator/debugger:

```
: RESET(RET)
:
```

4.45 ROUND_MODE

ROUND_MODE	Specifies and displays floating-point rounding mode
RM	

Format Set: ROUND_MODEΔ{Z|N}(RET)
Display: ROUND_MODE (RET)

Parameter • Rounding mode {Z|N}

Z:	Rounds toward zero (default).
N:	Rounds to the nearest value.

Function Specifies the floating-point rounding mode.

Examples (1) To display the current rounding mode:

```
: ROUND_MODE(RET)
ROUND TO ZERO
:
```

(2) To specify the round-to-nearest mode:

```
: ROUND_MODE N(RET)
:
```

4.46 SAVE_STATUS

SAVE_STATUS	Saves the current simulator/debugger status in a file
SS	

Format SAVE_STATUS[Δ<file name>](RET)

Parameter • <file name>
Specifies the name of the file in which the simulation status is saved.
When the file name is omitted, the file sdsh.sav is assumed.
When the file extension is omitted, the extension .sav is added.

Function Saves the current simulation status in a file.
- Name of the program file which has been loaded last
- CPU information
- Default radix
- Memory map setting information
- Trace condition
- Break information
- Register information

Example To save the simulation state in file test1.sav:

```
: SAVE_STATUS test1.sav(RET)
:
```

4.47 STACK_ANALYSIS

STACK_ANALYSIS	Enables/disables or resets the stack use analysis results
SA	

Format STACK_ANALYSISΔ{E|D|R}(RET)

Parameter • Enable/disable/reset {E|D|R}

E (enable):	Enables stack use analysis.
D (disable):	Disables stack use analysis.
R (reset):	Resets the stack use analysis results.

Function Enables/disables stack use analysis or resets the analysis results.

Description By resetting, only the analysis results are reset and the stack use analysis enable/disable setting is not changed.

Examples (1) To enable stack use analysis:

```
: STACK_ANALYSIS E(RET)
:
```

(2) To reset the stack use analysis results:

```
: STACK_ANALYSIS R(RET)
:
```

(3) To disable stack use analysis:

```
: STACK_ANALYSIS D(RET)
:
```

4.48 STACK_ANALYSIS_DISPLAY

STACK_ANALYSIS_DISPLAY	Displays the stack use analysis results
SD	

Format STACK_ANALYSIS_DISPLAY[ΔPC][ΔM](RET)

Parameters
- PC: Displays the program counter value (not displayed as a default).
- M: Displays the maximum/minimum stack pointer values (not displayed as a default).

Function Displays the cycle count and stack value when a stack has changed as the results of stack use analysis. When PC is specified, the program counter value when a stack has changed is displayed. When M is specified, the maximum and minimum stack values are displayed.

Description
(1) Analyzes stack use.
(2) Saves changes of the SP values during program execution.
The buffer is configured in the form of a ring and stores up to 9999 data.
When 10000 or more data is saved, the buffer is overwritten from its head.
(3) Displayed items are as follows:

```
CYCLE      SP       PC
XXXXXXXXXX XXXXXXXX XXXXXXXX
     :         :        :
XXXXXXXXXX XXXXXXXX XXXXXXXX
    CYCLE      SP       PC
Max XXXXXXXXXX XXXXXXXX XXXXXXXX
Min XXXXXXXXXX XXXXXXXX XXXXXXXX
```

[CYCLE]:	Cycle count
[SP]:	Stack pointer value
[PC]:	Program counter value
[Max]:	Displays the maximum stack pointer value.
[Min]:	Displays the minimum stack pointer value.

(4) Stack use analysis is disabled when the simulator is initiated.

Examples (1) To display the stack use analysis results:

```
: STACK_ANALYSIS_DISPLAY(RET)
CYCLE       SP
       333  00000FF0
     10000  00000900
    999998  00000FFC
:
```

(2) To display the stack use analysis results including the maximum and minimum program counter and stack pointer values:

```
: STACK_ANALYSIS_DISPLAY PC M(RET)
CYCLE       SP         PC
       333  00000FF0 000000F0
     10000  00000090 00000F00
9999999999  00000FFC 00000FF0
    CYCLE       SP       PC
Max 9999999999  00000FFC 00000FF0
Min      10000  00000090 00000F00
:
```

4.49 STATUS

STATUS	Displays the simulator/debugger status
ST	

Format STATUS(RET)

Function Displays the CPU type and endian when the simulator/debugger is started, and the execution cycle count and cache hit ratio when the simulator/debugger stops.

Display for SH-1, SH-2, SH-2E, and SH-DSP

```
CPU=xxx   ENDIAN=xxxx   CYCLE=xxxx
    (1)            (2)            (3)
```

Display for SH-3 and SH-3E

```
CPU=xxx   ENDIAN=xxxx   CYCLE=xxxx
    (1)            (2)            (3)
CACHE HIT=xx%
         (4)
```

Display for SH-4

```
CPU=xxx   ENDIAN=xxxx   CYCLE=xxxx
    (1)            (2)            (3)
INSTRUCTION CACHE HIT=xx%   OPERAND CACHE HIT=xx%
                     (5)                      (6)
```

(1) CPU type
(2) Endian
(3) Execution cycle count (10-digit decimal)
(4) Cache hit ratio (percentage)
(5) Instruction cache hit ratio (percentage)
(6) Operand cache hit ratio (percentage)

Examples (1) To display the simulator/debugger status when the CPU type SH-1 and big endian are specified at startup.

```
: STATUS (RET)
CPU=SH1   ENDIAN=BIG   CYCLE=128
:
```

(2) To display the simulator/debugger status when the CPU type SH-3E and big endian are specified at startup.

```
: STATUS (RET)
CPU=SH3E   ENDIAN=BIG   CYCLE=4186
CACHE HIT=89%
:
```

(3) To display the simulator/debugger status when the CPU type SH-4 and little endian are specified at startup.

```
: STATUS (RET)
CPU=SH4   ENDIAN=LITTLE   CYCLE=157353
INSTRUCTION CACHE HIT=89%   OPERAND CACHE HIT=34%
:
```

4.50 STEP

STEP	Performs step execution (executes subroutine as one step)
S	

Format STEP[Δ<step count>][ΔR](RET)

Parameters
- <step count>
  Specifies the number of instruction execution steps. (H'1 to H'FFFF)
  When omitted, one step is executed.
- Register content display R
  R (register): Displays the contents of the registers after instruction execution.

Function Executes instructions one at a time starting at the current program counter for the specified number of steps.

Description
(1) Displays the mnemonic of the executed instruction each time an instruction is executed.
If the R option is specified, the contents of the register after instruction execution is also displayed.
(2) For a subroutine branched by the BSR, JSR or BSRF instruction, execution is performed from the start of the subroutine to the instruction following the RTS instruction (since RTS is a delay branch instruction) as one step.
(3) Execution stops when the break condition set by a break command is satisfied or an error is detected by the simulator/debugger.
In this case, the stop cause is displayed.

Example To execute five instructions, executing the subroutine as though it were a single instruction:

```
: STEP 5 (RET)
00000000   MOV.L     R3,@R14
00000002   MOV.L     @(H'0084:8,PC), R1
00000004   JSR       @R1
00000006   NOP
00000008   LDS.L     @R15+, PR
+++ 5000 : Step normal end
:
```

4.51 STEP_G

STEP_G	Specifies step execution address range executing subroutine as one step
SG	

Format STEP_GΔ<start PC address>Δ<end PC address>[ΔR](RET)

Parameters
- <start PC address>
  Specifies the dummy start address (H'0 to H'FFFFFFFF).
  In this simulator/debugger, the start address of the step execution range is always the current PC value regardless of this setting.
- <end PC address>
  Specifies the end PC address of the step execution range.
- Register content display R
  R (register): Displays the contents of the register after step execution.

Function Executes instructions one at a time from the current PC address to the end PC address.

Description
(1) Displays the mnemonic of the last executed instruction.
If the R option is specified, the contents of the register after instruction execution is also displayed.
(2) For a subroutine branched by the BSR, JSR or BSRF instruction, execution is performed from the start of the subroutine to the instruction following the RTS instruction (since RTS is a delay branch instruction) as one step.
(3) Execution stops when the break condition set by a break command is satisfied or an error is detected by the simulator/debugger.
In this case, the stop cause is displayed.
(4) The relationship between the start and end PC addresses is as follows:
When the end PC address is an odd value, the last bit is set to 0 to make it an even value (ANDing with 0xfffffffe to make it an even value).
Examples: 1 → 0, 101 → 100, ffff → fffe

When current PC address = end PC address, step execution is performed at the current PC address by one step.
When current PC address > end PC address, step execution is performed at the current PC address by one step.
When current PC address < end PC address, step execution is performed from the current PC address to the end PC address.

Step execution stops when the PC value becomes outside the range from the current PC addresses and the end PC address (except during subroutine execution).

Example To perform step execution from the current PC address to address H'8:

```
: STEP_G 0 8 (RET)
00000008    LDS.L     @R15+,PR
+++ 5000 : Step normal end
:
```

4.52 STEP_INTO

STEP_INTO	Performs step execution
ST	

Format STEP_INTO[Δ<step count>][ΔR](RET)

Parameters
- <step count>
  Specifies the number of instruction execution steps. (H'1 to H'FFFF)
  When omitted, one step is executed.
- Register content display R
  R (register): Displays the contents of the registers after instruction execution.

Function Executes instructions one at a time starting at the current program counter for the specified number of steps.
When a subroutine is called by the program, the called subroutine is also executed one step at a time.

Description
(1) Displays the mnemonic of the executed instruction each time an instruction is executed.
If the R option is specified, the contents of the register after instruction execution is also displayed.
(2) Execution stops when the break condition set by a break command is satisfied or an error is detected by the simulator/debugger.
In this case, the stop cause is displayed.

Examples (1) To execute one instruction and then display the mnemonic of the executed instruction and the contents of the registers following the instruction execution (For the SH-3):

```
: STEP_INTO R(RET)
PC=00001002 SR=700000F0:-MRB-------------------IIII---- SP=00000000
GBR=00000000 VBR=00000000 MACH=00000000 MACL=00000000 PR=00000000
R0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R0_BANK-R3-BANK 00000000 00000000 00000000 00000000
R4_BANK-R7-BANK 00000000 00000000 00000000 00000000
SSR=00000000 SPC=00000000
PTEH=00000000 PTEL=00000000 TTB=00000000 TEA=00000000 MMUCR=00000000
EXPEVT=00000000 INTEVT=00000000 TRA=00000000 CCR=00000000
00001000 MOV      #H'00,R3
+++5000 : Step normal end
:
```

(2) To execute three instructions:

```
: STEP_INTO 3(RET)
00000404  MOV.L      #0000002E,R4
00000406  MOV.L      #FFFFFFFF,R3
00000408  ADD.L      R1,R2
+++5000 : Step normal end
:
```

4.53 STEP_INTO_G

STEP_INTO_G	Specifies step execution range
SIG	

Format STEP_INTO_GΔ<start PC address>Δ<end PC address>[ΔR](RET)

Parameters
- <start PC address>
  Specifies the dummy start address (H'0 to H'FFFFFFFF).
  In this simulator/debugger, the start address of the step execution range is always the current PC value regardless of this setting.
- <end PC address>
  Specifies the end PC address of the step execution range.
- Register content display R
  R (register): Displays the contents of the register after instruction execution.

Function Executes instructions one at a time from the current PC address to the end PC address. When a subroutine is called by the program, the called subroutine is also executed one step at a time.

Description
(1) Displays the mnemonic of the instruction which has been executed last.
If the R option is specified, the contents of the register after instruction execution is also displayed.
(2) When a subroutine is called by the program, the called subroutine is also executed one step at a time.
(3) Execution stops when the break condition set by a break command is satisfied or an error is detected by the simulator/debugger.
In this case, the stop cause is displayed.
(4) The relationship between the start and end PC addresses is as follows:
When the end PC address is an odd value, the last bit is set to 0 to make it an even value (ANDing with 0xfffffffe to make it an even value).
Examples: 1 → 0, 101 → 100, ffff → fffe

When current PC address = end PC address, step execution is performed at the current PC address by one step.
When current PC address > end PC address, step execution is performed at the current PC address by one step.
When current PC address < end PC address, step execution is performed from the current PC address to the end PC address.

Step execution stops when the PC value becomes outside the range from the current PC addresses and the end PC address (also during subroutine execution).

Example To perform step execution from the current PC address to address H'6:

```
: STEP_INTO_G 0 6(RET)
00000006   ADD.L      R1,R2
+++5000 : Step normal end
:
```

4.54 TLB (Only for the SH-3/SH-3E/SH-4 Series)

TLB	Modifies the TLB contents
TLB	

Format
- For SH-3 and SH3E
  Modification: TLBΔ<index>Δ<way>[Δ<AA data>][Δ:Δ<DA data>](RET)
  Interactive mode: TLBΔ<index>Δ<way>(RET)
- For SH-4
  Modification: TLB[Δ{I|U}]Δ<entry>[Δ<AA data>][Δ:Δ<DA data>](RET)
  Interactive mode: TLB[Δ{I|U}]Δ<entry>(RET)

Parameters
- <index>
  Specifies the TLB way index to be modified. (H'00 to H'1F)
- <way>
  Specifies the TLB way to be modified. (H'0 to H'3)
- <AA data>
  Specifies data to be written into the address array.
- <DA data>
  Specifies data to be written into the data array.
- TLB type to be modified {I|U}
  I: Specifies the instruction TLB (ITLB) (default).
  U: Specifies the unified TLB (UTLB).
- <entry>
  Specifies the TLB entry to be modified.

Function Modifies the TLB contents.

Description (1) Modification (direct):
Modifies the TLB contents with the specified data.

(2) Modification (interactive mode):
If the modification data (AA and DA data) is omitted, the TLB contents are modified interactively using the following formats.
Displays the current data, and requests the modification data to be input.

For SH-3 and SH-3E

```
: TLBΔ<index>Δ<way>(RET)
ii w aaaaaaaa/dddddddd : [Δ<AA data>][Δ:Δ<DA data>](RET)
ii w aaaaaaaa/dddddddd :
(1)(2)      (3)           (4)
```

(1): Index (2-digit hexadecimal)
(2): Way (1-digit hexadecimal)
(3): Current address array data (8-digit hexadecimal)
(4): Current data array data (8-digit hexadecimal)

For SH-4

```
: TLBΔ<entry>(RET)
ee aaaaaaaa/dddddddd : [Δ<AA data>][Δ:Δ<DA data>](RET)
ee aaaaaaaa/dddddddd :
(1)     (2)          (3)
```

(1): Entry (2-digit hexadecimal)
(2): Current address array data (8-digit hexadecimal)
(3): Current data array data (8-digit hexadecimal)

The following can be entered instead of modification data:

.(period):	Terminates the TLB command.
^:	Returns to the previous TLB entry.
(RET) only:	Goes to the next TLB entry.

Examples (1) To modify the entry in index 0 and way 0 in SH-3:

```
: TLB 0 0 00000000 : 00000000(RET)
:
```

(2) To modify the TLB contents sequentially from the entry in index 0 and way 0 in SH-3:

```
: TLB 0 0(RET)
00 0 00000000/00000000 : 00000101 : 00000500 (RET)
00 1 00000000/00000000 : 00000101 : 00000900 (RET)
00 2 00000000/00000000 : (RET)
00 3 00000000/00000000 : ^(RET)
00 2 00000000/00000000 : 00000101 : 00001100 (RET)
00 3 00000000/00000000 : 00000101 : 00001100 (RET)
01 0 00000000/00000000 : .(RET)
:
```

4.55 TLB_DUMP (Only for the SH-3/SH-3E/SH-4 Series)

TLB_DUMP	Displays the TLB contents
TLBD	

Format
- For SH-3 and SH3E
  TLB_DUMP(RET)
- For SH-4
  TLB_DUMP[{ΔI|U}](RET)

Parameter
- TLB type to be displayed {I|U}
  I: Specifies the instruction TLB (ITLB) (default).
  U: Specifies the unified TLB (UTLB).

Function Displays the contents of the TLB address and data arrays.

Description Displays the TLB contents in the following formats:

For SH-3 and SH-3E

```
<NO>      <WAY0>            <WAY1>            <WAY2>            <WAY3>
 ii aaaaaaaa/dddddddd aaaaaaaa/dddddddd aaaaaaaa/dddddddd aaaaaaaa/dddddddd
(1)      (2)      (3)      (2)      (3)      (2)      (3)      (2)      (3)
```

(1): Index (2-digit hexadecimal)
(2): Current address array data (8-digit hexadecimal)
Bits 16 to 12 (five bits) are always 0.
(3): Current data array data (8-digit hexadecimal)

For SH-4

```
<NO>  <ADDR ARRAY> <DATA ARRAY>
 ee       aaaaaaaa/dddddddd
 (1)          (2)      (3)
```

(1): Entry (2-digit hexadecimal)
(2): Address array data (8-digit hexadecimal)
(3): Data array data (8-digit hexadecimal)

Example To display the contents of all indexes in SH-3:

```
: TLB_DUMP(RET)
<NO>       <WAY0>             <WAY1>             <WAY2>             <WAY3>
 00  00000000/00000000 00000000/00000000 00000000/00000000 00000000/00000000
 01  00000000/00000000 00000000/00000000 00000000/00000000 00000000/00000000
 :           :                  :                  :                  :
 1F  00000000/00000000 00000000/00000000 00000000/00000000 00000000/00000000
:
```

4.56 TLB_FLUSH (Only for the SH-3/SH-3E/SH-4 Series)

TLB_FLUSH	Flushes the TLB contents
TLBF	

Format
- SH-3/SH-3E series
  TLB_FLUSH(RET)
- SH-4 series
  TLB_FLUSH[{∆I|U}](RET)

Parameters
- Type of TLB to flush {I|U}
  I: Specifies instruction TLB (ITLB). (Default)
  U: Specifies unified TLB (UTLB).

Function Flushes the TLB contents.

Example To flush the TLB contents in SH3:

```
: TLB_FLUSH(RET)
:
```

4.57 TLB_SEARCH (Only for the SH-3/SH-3E/SH-4 Series)

TLB_SEARCH	Searches for the TLB contents
TLBS	

Format
- SH-3/SH-3E series
  TLB_SEARCHΔ<address>[Δ<address type>](RET)
- SH-4 series
  TLB_SEARCH[Δ{I|U}]Δ<address>[Δ<address type>](RET)

Parameters
- <address>
  Specifies the address to be searched for.
- <address type>
  Specifies the type of address to be searched for as follows.
  P: Physical address
  V: Virtual address (default)
- Type of TLB to search {I|U}
  I: Specifies instruction TLB (ITLB). (Default)
  U: Specifies unified TLB (UTLB).

Function Searches for the TLB address array which holds the specified virtual address, or Searches for the TLB data array which holds the specified physical address.
The index, way number, address and data array values of the corresponding entry are displayed.

Examples (1) To search for the TLB contents using virtual address H'00000000 in SH-3:

```
: TLB_SEARCH 0 (RET)
00 0 00000101/20000158
00 2 00000103/20100158
:
```

(2) To search for the TLB contents using physical address H'20000000 in SH-3:

```
: TLB_SEARCH 20000000 P (RET)
00 0 00000101/20000158
:
```

4.58 TRACE

TRACE	Displays trace buffer
T	

Format TRACE[Δ<offset>[Δ<count>]](RET)

Parameters
- <offset>
  Specifies the first cycle to be displayed (0 to 1023).
  Indicates the number of cycles from the head of the trace buffer.
  Unless specified, the ninth and later cycles from the last are displayed.
- <count>
  Specifies the count to be displayed (1 to 1024).
  Unless specified, ten cycles are displayed.
  When specifying the count, the offset must also be specified.

Function Displays the contents of the trace buffer.
The last cycle (which has been executed last) of the buffer is cycle 0 and the previous cycles have negative values.

Description (1) The contents to display are as follows:
SH-1, SH-2, SH-DSP, and SH-2E series:

```
PTR   CYCLE    ADDR-BUS PIPELINE       INSTRUCTION
XXXXX XXXXXXXX XXXXXXXX XXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

[PTR]:	Pointer in the trace buffer (the instruction which has been executed last is "0")
[CYCLE]:	Accumulative cycle count of executed instructions (cleared by pipeline reset)
[ADDR-BUS]:	Instruction address
[PIPELINE]:	The meanings of the pipeline execution state symbols are as follows:

F:	Instruction fetch (with memory access)
f:	Instruction fetch (with no memory access)
D:	Instruction decode
E:	Instruction execution
M:	Memory access
W:	Write back
P:	DSP (SH-DSP series only)
m:	Multiplier execution
-:	Stall inherent to instruction
>:	Split
<:	Stall due to contention

For details on pipeline operation, refer to the programming manual of each device.

[INSTRUCTION]: Mnemonic instruction and data access (indicated in the form of "transfer destination <– transfer data")

SH-3 and SH-3E series:

```
PTR   CYCLE      ADDR-BUS  DATA-BUS CODE NO INSTRUCTION
IF DE EX MA SW ACCESS DATA

XXXXX XXXXXXXXXX XXXXXXXX  XXXXXXXX XXXX XX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX XX XX XX XX XXXXXXXXXXXXXXXXXXXXXXXXXXX
```

[PTR]:	Pointer in the trace buffer (the instruction which has been executed last is "0")
[CYCLE]:	Accumulative cycle count of executed instructions (cleared by pipeline reset)
[ADDR-BUS]:	Data on the address bus
[DATA-BUS]:	Data on the data bus
[CODE]:	Instruction code
[No]:	Instruction execution number (corresponding to the execution number of each stage)
[INSTRUCTION]:	Mnemonic instruction
[IF]:	Execution number of fetched instruction
[DE]:	Execution number of decoded instruction
[EX]:	Execution number of executed instruction
[MA]:	Execution number of instruction that accessed memory
[SW]:	Execution number of instruction that wrote back data
[ACCESS DATA]:	Contents of data access (indicated in the form of "transfer destination <– transfer data")

SH-4 series:

```
PTR   CYCLE      ADDRESS code1 code2 EX-EAS LS-EAS BR-EAS FP-EXASD
INSTRUCTION       ACCESS DATA
XXXXX XXXXXXXXXX XXXXXXXX XXXX  XXXX X X X  X X X  X X X  X X X X X
XXXXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXXXXXXXXXXXXXX
```

[PTR]:	Pointer in the trace buffer (the instruction which has been executed last is "0")
[CYCLE]:	Accumulative cycle count of executed instructions (cleared by pipeline reset)
[ADDRESS]:	Program counter address
[code1]:	Code1 of the fetched program
[code2]:	Code2 of the fetched program
[EX-EAS]:	Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by EX pipeline execution
[LS-EAS]:	Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by LS pipeline execution
[BR-EAS]:	Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by BR pipeline execution
[FP-EXASD]:	Number of the instruction that has been executed (E), accessed memory (A), or wrote back data (S) by FP pipeline execution (The X stage uses only FSCA, FSRRA, FIPR, and FTRV instructions, and the D stage uses only FDIV and FSQRT instructions)
[INSTRUCTION]:	Instruction number assigned to the instruction to be executed, Memory address, instruction code, and mnemonic of the instruction to be executed.
[ACCESS DATA]:	Contents of data access (indicated in the form of "transfer destination <– transfer data")

Note Trace information is displayed for up to 1023 instructions.

Example To display five cycles from the head of the trace buffer:

SH-1, SH-2 and SH-DSP series:

```
: TRACE 0 5(RET)
PTR   CYCLE      ADDR-BUS PIPELINE  INSTRUCTION
-1023 0000010193 000001B0 FFDE>MM   :MOV.L  R1,@R6    000001EC <- 00000001
-1022 0000010195 000001B2 fD>E>     :TST    R5,R5     T<-(0)
-1021 0000010197 000001B4 FFD>E>    :BT     000001BA  T(0)
-1020 0000010199 000001B6 f>D>E     :MOV    #FF,R2    R2 <- FFFFFFFF
-1019 0000010200 000001B8 FFDE>MM   :MOV.L  R2,@R6    000001EC <- FFFFFFFF
:
```

SH-3 and SH-3E series:

```
: TRACE 0 5(RET)
PTR   CYCLE      ADDR-BUS DATA-BUS CODE NO INSTRUCTION         IF DE EX MA SW
ACCESS DATA
-0014 0000000000 BEADCAFE 00000000 0009 -- ------------------- -- -- -- -- --
-0013 0000000001 00000000 00000000 0009 -- ------------------- 01 -- -- 00 --
-0012 0000000002 00000002 4F227FC8 4F22 01 STS.L PR,@-R15      02 01 -- -- --
-0011 0000000003 00000004 4F227FC8 7FC8 02 ADD #C8,R15         03 02 01 -- --
-0010 0000000004 00001F84 E3001F32 E300 03 MOV #00000000,R3    04 03 02 01 --
(01):00001F84 <- 00000000
:
```

SH-2E series:

```
: t 0 5(RET)
PTR   CYCLE      ADDR-BUS PIPELINE          INSTRUCTION
-0010  0000000000                           :PIPELINE RESET
-0009 0000000002 80000000 FFDE>             :ADD    #FC, R15  R15<-00000FEC
-0008 0000000004 80000002 fD>EMM>>          :MOV.L  R4, @R15  00000FEC<-00000000
-0007 0000000007 80000004 FFD<<E>           :MOV    #0A, R2  R2<-0000000A
-0006 0000000009 80000006 f<<D>EMM>>        :MOV.L  R2, @R15  00000FEC<-0000000A
:
```

SH-4 series:

```
: t 0 5(RET)
PTR   CYCLE      ADDRESS  code1 code2 EX-EAS LS-EAS BR-EAS FP-EXASD  INSTRUCTION
-1023 0000004426 00000060 xxxx  xxxx  x x x  x x x  x x x  x x x x x
-1022 0000004427 00000060 xxxx  xxxx  x x x  x x x  x x x  x x x x x
-1021 0000004428 00000064 xxxx *E30A  x x x  x x x  x x x  x x x x x [5
(0000005E): E30A  MOV       #0A, R3]
-1020 0000004429 00000064 xxxx  xxxx  5 x x  x x x  x x x  x x x x x
-1019 0000004430 00000064 xxxx  xxxx  x 5 x  x x x  x x x  x x x x x
:
```

4.59 TRACE_CONDITION

TRACE_CONDITION	Sets trace condition, and starts or stops trace
TC	

Format Start: TRACE_CONDITION[[Δ{I|S}][ΔE][Δ{C|B}]](RET)
Stop: TRACE_CONDITIONΔD(RET)

Parameter
- Instruction type {I|S}

I (instruction):	All instructions are saved in the trace buffer (default).
S (subroutine):	Only subroutine calling instructions (BSR, JSR, and BSRF) are saved in the trace buffer.

- Trace start/stop {E|D}

E (enable):	Starts saving in the trace buffer (default).
D (disable):	Terminates saving in the trace buffer.

- Trace buffer full handling {C|B}

C (continue):	Overwrites the previous contents of the trace buffer after the trace buffer overflows. (default)
B (break):	Interrupts program execution when the trace buffer overflows.

Function Sets the trace condition.

Description Saving in the trace buffer has been disabled when the simulator is initiated.

Examples (1) To save all instructions in the trace buffer following the execution of the command:

```
: TRACE_CONDITION I (RET)
:
```

(2) To save only subroutine calls in the trace buffer:

```
: TRACE_CONDITION S E(RET)
:
```

(3) To terminate saving in the trace buffer:

```
: TRACE_CONDITION D(RET)
:
```

4.60 TRACE_CLEAR

TRACE_CLEAR	Clears trace buffer
TL	

Format TRACE_CLEAR(RET)

Function Clears the contents of the trace buffer.

Description Only the trace buffer is cleared and the trace conditions (the instruction type, the trace start/end setting and processing when the trace buffer is full) are not changed.

Example To clear the contents of the trace buffer:

```
: TRACE_CLEAR(RET)
:
```

4.61 TRAP_ADDRESS

TRAP_ADDRESS	Sets the system call start address
TA	

Format TRAP_ADDRESSΔ<start address>(RET)

Parameter • <start address>
Specifies the system call start address.
After the start address is specified, the system call becomes valid.

Function Sets the system call start address for inputting and outputting characters between the user program and the standard I/O device, and file I/O. Only one address can be set.
If the branch address of an executed JSR, BSR, and BSRF instruction is the same as the address specified with this command, normal simulation is not performed, but rather the system call indicated by the function code is executed.
A parameter block and an I/O buffer must be allocated within the user program.
The user program must set up R0 and R1, the parameter block, and the I/O buffer before executing the JSR, BSR, or BSRF instruction.
Simulation is restarted from the instruction following the JSR, BSR, or BSRF instruction after the system call processing.
The contents of R0 and R1 and the other registers are shown below.
Since the contents stored in the parameter block differ for each system call function, the parameter block contents are described under each function.

	MSB 1 byte	1 byte		LSB
R0 register	H'01	Function code	–	–
R1 register	Parameter block address			

Notes <System Call Functions>
The simulator/debugger provides functions to issue system calls to the host computer from the user program.
The following table lists the system calls that can be used by a user program.

Table 4.3 System Call Functions

Function Code	Function	Description
H'21	GETC	Inputs one character from standard input
H'22	PUTC	Outputs one character to standard output
H'23	GETS	Inputs a line of characters from standard input
H'24	PUTS	Outputs a line of characters to standard output
H'25	FOPEN	Opens a file
H'06	FCLOSE	Closes a file
H'27	FGETC	Inputs one byte from a file
H'28	FPUTC	Outputs one byte to a file
H'29	FGETS	Inputs a line from a file
H'2A	FPUTS	Outputs a line to a file
H'0B	FEOF	Checks for end of file
H'0C	FSEEK	Moves the file pointer
H'0D	FTELL	Returns the current position of the file pointer

(1) GETC

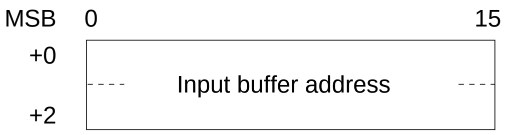
<Function>

Inputs one character from standard input.

<Function code>

H'21

<Parameter block>



<Example>

To input one character from standard input (usually the keyboard):

```
         MOV.L PAR_ADR,R1
         MOV.L REQ_COD,R0
         MOV.L CALL_ADR,R3
         JSR   @R3
         NOP
STOP     NOP
SYS_CALL NOP
         .ALIGN 4
CALL_ADR .DATA.L SYS_CALL
REQ_COD  .DATA.L H'01210000
PAR_ADR  .DATA.L PARM
PARM     .DATA.L INBUF
INBUF    .RES.B 2
         .END
```

(2) PUTC

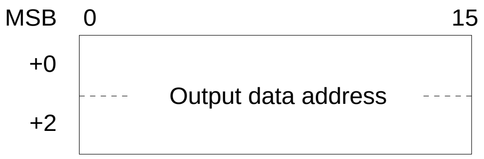
<Function>

Outputs one character to standard output.

<Function code>

H'22

<Parameter block>



<Example>

To output the character 'A' to standard output (usually the console):

```
          MOV.L PAR_ADR,R1
          MOV.L REQ_COD,R0
          MOV.L CALL_ADR,R3
          JSR   @R3
          NOP
STOP      NOP
SYS_CALL  NOP
          .ALIGN 4
CALL_ADR  .DATA.L SYS_CALL
REQ_COD   .DATA.L H'01220000
PAR_ADR   .DATA.L PARM
PARM      .DATA.L OUTDATA
OUTDATA   .DATA.B "A"
          .END
```

(3) GETS

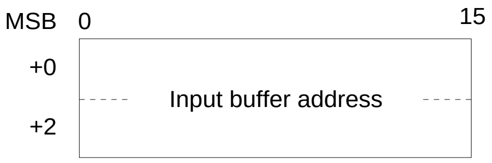
<Function>

Inputs a line of characters from standard input.
A line feed character (LF) terminates the input line.
Up to 79 characters can be input in a line.
If more than 79 characters are input, the 80th character will be converted to a line feed (LF).

<Function code>

H'23

<Parameter block>



<Example>

To input one line from standard input (usually the keyboard):

```
         MOV.L PAR_ADR,R1
         MOV.L REQ_COD,R0
         MOV.L CALL_ADR,R3
         JSR   @R3
         NOP
STOP     NOP
SYS_CALL NOP
         .ALIGN 4
CALL_ADR .DATA.L SYS_CALL
REQ_COD  .DATA.L H'01230000
PAR_ADR  .DATA.L PARM
PARM     .DATA.L INBUF
INBUF    .RES.B 80
         .END
```

(4) PUTS

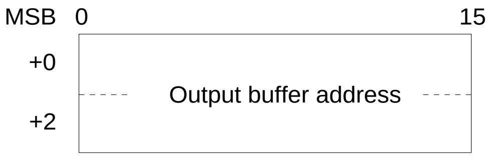
<Function>

Outputs a line of characters to standard output.
A line feed character (LF) terminates the output line.
Up to 131 characters can be output on a line.
If more than 131 characters are specified, the 132nd character will be converted to a line feed (LF).

<Function code>

H'24

<Parameter block>



<Example>

To output the string "Hello world" to standard output (usually the console):

```
            MOV.L PAR_ADR,R1
            MOV.L REQ_COD,R0
            MOV.L CALL_ADR,R3
            JSR   @R3
            NOP
  STOP      NOP
  SYS_CALL  NOP
            .ALIGN 4
  CALL_ADR  .DATA.L SYS_CALL
  REQ_COD   .DATA.L H'01240000
  PAR_ADR   .DATA.L PARM
  PARM      .DATA.L OUTDATA
  OUTDATA   .SDATA."Hello world"
            .DATA.B H'0A
            .END
```

(5) FOPEN

<Function>

The FOPEN opens a file and returns the file number.

After this processing, the returned file number must be used to input, output, or close files.

A maximum of 256 files can be open at the same time.

<Function code>

H'25

<Parameter block>

MSB	0 One byte	One byte 15
+0	Return value	File number
+2	Open mode	Unused
+4	Start address of file name	
+6		

<Parameters>

- Return value (output)

  0: Normal completion

  –1: Error
- File number (output)

  The number to be used in all processing after opening.
- Open mode (input)

  H'00: "r"

  H'01: "w"

  H'02: "a"

  H'03: "r+"

  H'04: "w+"

  H'05: "a+"

  H'10: "rb"

  H'11: "wb"

  H'12: "ab"

  H'13: "r+b"

  H'14: "w+b"

  H'15: "a+b"

  These modes are interpreted as follows.

  "r": Open for reading.

  "w": Open an empty file for writing.

  "a": Open for appending (write starting at the end of the file).

  "r+": Open for reading and writing.

  "w+": Open an empty file for reading and writing.

"a+": Open for reading and appending.

"b": Open in binary mode.

- Start address of file name (input)

  The start address of the area for storing the file name.

(6) FCLOSE

<Function>

Closes a file

<Function code>

H'06

<Parameter block>

MSB	0 One byte	One byte 15
+0	Return value	File number

<Parameters>

- Return value (output)

  0: Normal completion

  –1: Error

- File number (input)

  The number returned when the file was opened.

(7) FGETC

<Function>

Inputs one byte from a file

<Function code>

H'27

<Parameter block>

MSB	0 One byte	One byte 15
+0	Return value	File number
+2	Unused	
+4	Start address of input buffer	
+6		

<Parameters>

- Return value (output)
  0: Normal completion
  –1: EOF detected or error
- File number (input)
  The number returned when the file was opened.
- Start address of input buffer (input)
  The start address of the buffer for storing input data.

(8) FPUTC

<Function>

Outputs one byte to a file

<Function code>

H'28

<Parameter block>

MSB	0	15
	One byte	One byte
+0	Return value	File number
+2	Unused	
+4	Start address of output buffer	
+6		

<Parameters>

- Return value (output)
  0: Normal completion
  –1: Error
- File number (input)
  The number returned when the file was opened.
- Start address of output buffer (input)
  The start address of the buffer used for storing the output data.

(9) FGETS

<Function>

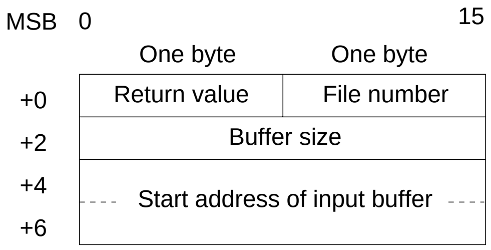
Reads character string data from a file.

Data is read until either a new line code or a NULL code is read, or until the buffer is full.

<Function code>

H'29

<Parameter block>



<Parameters>

- Return value (output)
  0: Normal completion
  –1: EOF detected or error
- File number (input)
  The number returned when the file was opened.
- Buffer size (input)
  The size of the area for storing the read data. A maximum of 256 bytes can be stored.
- Start address of input buffer (input)
  The start address of the buffer for storing input data.

(10) FPUTS

<Function>

Writes character string data to a file.

The NULL code that terminates the character string is not written to the file.

<Function code>

H'2A

<Parameter block>

MSB 0 15

	One byte	One byte
+0	Return value	File number
+2	Unused	
+4	---- Start address of output buffer ----	
+6		

<Parameters>

- Return value (output)
  0: Normal completion
  –1: Error
- File number (input)
  The number returned when the file was opened.
- Start address of output buffer (input)
  The start address of the buffer for storing output data.

(11) FEOF

<Function>

Checks for end of file

<Function code>

H'0B

<Parameter block>

MSB	0 One byte	One byte 15
+0	Return value	File number

<Parameters>

- Return value (output)

  0: File pointer is not at EOF

  –1: EOF detected

- File number (input)

  The number returned when the file was opened.

(12) FSEEK

<Function>

Moves the file pointer to the specified position

<Function code>

H'0C

<Parameter block>

MSB 0 15

	One byte	One byte
+0	Return value	File number
+2	Direction	Unused
+4	Offset	
+6		

<Parameters>

• Return value (output)

0: Normal completion

–1: Error

• File number (input)

The number returned when the file was opened.

• Direction (input)

0: The offset specifies the position as a byte count from the start of the file.

1: The offset specifies the position as a byte count from the current file pointer.

2: The offset specifies the position as a byte count from the end of the file.

• Offset (input)

The byte count from the location specified by the direction parameter.

(13) FTELL

<Function>

Returns the current position of the file pointer

<Function code>

H'0D

<Parameter block>

MSB 0 15

	One byte	One byte
+0	Return value	File number
+2	Unused	
+4	Offset	
+6		

<Parameters>

- Return value (output)
  0: Normal completion
  –1: Error
- File number (input)
  The number returned when the file was opened.
- Offset (input)
  The current position of the file pointer, as a byte count from the start of the file.

Example Set the system call address to H'10:

```
:  TRAP_ADDRESS 10(RET)
:
```

4.62 TRAP_ADDRESS_DISPLAY

TRAP_ADDRESS_DISPLAY	Displays the system call start address
TD	

Format TRAP_ADDRESS_DISPLAY(RET)

Function Displays the system call start address and whether the call is enabled or disabled.

Description The system call start address and whether the call is enabled or disabled are displayed in the following format:

```
: TD(RET)
  aaaaaaaa b
```

aaaaaaaa:	System call start address
b:	E (enable) = System call enabled
	D (disable) = System call disabled

Example To display the system call start address and whether the call is enabled or disabled:

```
: TRAP_ADDRESS_DISPLAY(RET)
00000010 E
:
```

4.63 TRAP_ADDRESS_ENABLE

TRAP_ADDRESS_ENABLE	Enables/disables the system call start address
TE	

Format TRAP_ADDRESS_ENABLEΔ{E|D}(RET)

Parameters
- <start address>
  Specifies the system call address.
  After the start address is specified, the system call becomes valid.
- Enable/disable {E|D}
  E (enable): Enables the set system call start address.
  D (disable): Disables the set system call start address.

Function Enables/disables the system call for inputting and outputting characters from the user program to the standard I/O device and file I/O.

Example To disable the system call start address:

```
: TRAP_ADDRESS_ENABLE D(RET)
:
```

4.64 .<register>

.<register>	Changes the contents of registers

Format General command form: .<register>Δ{<data>|<real number>}(RET)
Interactive mode: .<register>(RET)

Parameters
- <register>
  Specifies the general, control, system, management, DSP or FPU register name.
- <data>
  Specifies a new value.
- <real number>
  Specifies a new value in single-precision floating-point format ("F' " is added before the value) or double-precision floating-point format ("D' " is added before the value).

Function Changes the contents of registers to the specified value.

Description
(1) R0 to R15 can be specified for the general register name. SP can be specified instead of R15.
For the SH-3, SH-3E, and SH-4 series, R0_BANK to R7_BANK can also be specified.
Between R0 to R7 and R0_BANK to R7_BANK, the value change results are reflected to each other according to the contents of the MD and RB bits of SR.
(2) SR, GBR and VBR can be specified for the control register name.
For the SH-DSP series, RS, RE and MOD can also be specified.
For the SH-3, SH-3E, and SH-4 series, SSR and SPC can also be specified.
For the SH-4 series, SGR and DBR can also be specified.
(3) MACH, MACL, PR and PC can be specified for the system register name.
(4) FPUL, FPSCR and FR0 to FR15 can be specified for the FPU register name (for the SH-2E, SH-3E, and SH-4 series).
For the SH-4 series, XF0-XF15, DR0-DR14, and XD0 to XD14 can also be specified.
(5) PTEH, PTEL, TTB, TEA, MMUCR, EXPEVT, INTEVT, TRA and CCR can be specified for the management register name (only for the SH-3, SH-3E, and SH-4 series).
For the SH-4 series, QACR0 and QACR1 can also be specified.
(6) A0, A0G, A1, A1G, M0, M1, X0, X1, Y0, Y1 and DSR can be specified for the DSP register name (only for the SH-DSP series).
A value of up to eight bits can be set for the A0G and A1G registers.
When a value exceeding eight bits is specified, the lower eight bits become valid.

(7) After a command for specifying the interactive mode is input, the interactive mode is entered after the contents of the specified address is displayed.

```
.<register>(RET)
register data:[{<data>|<real number>|^}](RET)
register data:[{<data>|<real number>|^}](RET)
              .
              .
              .
register data: .(RET)
```

register data:	Displays the data before change.
<data>:	Specifies new data.
<real number>:	Specifies a new value in single-precision or double-precision floating-point format.
^:	Displays the contents of the previous register.
(RET) only:	Displays the contents of the next register.
. (period):	Terminates the .<register> command.

Examples (1) To change the contents of register R0 to H'1000:

```
: .R0 1000(RET)
:
```

(2) To change the contents of register R1 to H'9999:

```
: .R1 9999(RET)
:
```

(3) To change the contents of registers one by one from register R1 in interactive mode and display the contents of the previous and next registers:

```
: .R1(RET)
R1         00009999 : 5678(RET)
R2         00000000 : ^(RET)
R1         00005678 : (RET)
R2         00000000 : 345(RET)
R3         00000000 : ^(RET)
R2         00000345 : .(RET)
:
```

(4) To display and change the contents of registers from system register PC in interactive mode:

```
: .PC(RET)
PC        00000100 : 400(RET)
SR        00000000 : .(RET)
:
```

(5) To change the contents of FPU register FR0 to F'9876543e+21:

```
: .FR0 F'9876543e+21(RET)
:
```

4.65 Limitations

This section explains the limitations of using the SuperH^{TM} RISC engine simulator/debugger.

(1) When an operator is included in the input character string at command input, the numerals before and after the operator are calculated and used.

Note that calculation is not conducted if there is a space before or after an operator.

Floating-point data is not calculated.

Examples

```
: me 0 10+1 ;b
: md 0 10
address  +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000 11 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
:
```

(2) When displaying the mnemonic code of the instruction which has been executed last using a Go or Step command, other code than that of the last instruction may be displayed. This is because two instructions may be decoded in one step since one step unit has been set as from an E stage to the next E stage, displaying the code of the instruction which has been executed first.

(3) If a delayed branch instruction is included at the end of the memory area, a Memory Access Error occurs when the instruction is executed.

This is because a delayed branch instruction fetches the next instruction, but no memory is available for pre-fetching this instruction to be read and discarded, causing an error.

(4) When W, L, D or S is specified as the size for the Memory_Display command, specify the boundary of each size for the start address.

Otherwise, the memory contents are displayed as all Fs.

Examples

```
: md 0 10 ;w
address  +0   +2   +4   +6   +8   +A   +C   +E
00000000 0001 0002 0003 0004 0005 0006 0007 0008
: md 1 10 ;w
address  +0   +2   +4   +6   +8   +A   +C   +E
00000001 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
:
```

(5) The tables below show special floating-point values.
How to express them depends on the host computer.

Table 4.4 Special Value Expressions in Single-Precision

Value	SPARC	HP9000
0x00000000	0.000000e+00	0.000000e+00
0x80000000	-0.000000e+00	-0.000000e+00
0x7F800000	Inf	+.+00000e+01
0xFF800000	-Inf	--.-00000e+01
0x7F800001	NaN	?.00000e+00

Table 4.5 Special Value Expressions in Double-Precision

Value	SPARC	HP9000
0x0000000000000000	0.000000000000000e+00	0.000000000000000e+00
0x8000000000000000	-0.000000000000000e+00	-0.000000000000000e+00
0x7FF0000000000000	Infinity	+.+00000000000000e+01
0xFFF0000000000000	-Infinity	--.-00000000000000e+01
0x7FF8000000000001	NaN	?.000000000000000e+00

(6) The simulator/debugger for the HP9000 has the following limitations on floating-point operation.

- Quiet NAN (not-a number)
  While the result is a quiet NAN in the actual CPU, it may be a signaling NAN in the simulator/debugger.
- Invalid operation flag in the FPSCR
  Even when an invalid operation is performed, the invalid operation flag may not be set.

(7) Do not input a Tab code during inputting a command since it does not function as a delimiter.

Section 5 Message List

5.1 Information Messages

The simulator/debugger outputs information messages to notify the users of execution progress. Table 5.1 lists information messages output by the simulator/debugger.

Table 5.1 Information messages

Error No.	Message	Description
5000	Step normal end	Execution of STEP, STEP_G, STEP_INTO, STEP_INTO_G, or STEP_OUT command was completed normally.
5001	PC breakpoint	Execution was interrupted due to the occurrence of a breakpoint condition.
5002	Break sequence	Execution was interrupted due to the occurrence of a break sequence condition.
5003	Break data	Execution was interrupted due to the occurrence of a break data condition.
5004	Break register	Execution was interrupted due to the occurrence of a break register condition.
5005	Break access	Execution was interrupted due to the occurrence of a break access condition.
5006	Trace buffer full	Execution was interrupted since the Break mode had been selected by "Trace buffer full handling" in the Trace Acquisition dialog box and the trace buffer had become full.
5007	Sleep	Execution was interrupted due to the execution of a SLEEP instruction.
5008	Stop	Execution was interrupted by the [STOP] button.

5.2 Error Messages

The simulator/debugger outputs error messages to notify users of user program errors or operation errors. Table 5.2 lists the error messages.

Table 5.2 Error Messages

No.	Message	Contents/Measures
0001	Program error	An internal error occurred. Contact the nearest Hitachi office.
0007	Not enough memory	Memory to be used by the simulator/debugger cannot be allocated. Extend memory or change the user program.
0009	User aborted	Processing was interrupted by the break key input.
0010	File not found	The specified file was not found.
0011	Invalid CPU kind	The specified debug information file does not correspond to the CPU setting.
0501	Invalid parameter	The parameter value is invalid.
0502	Invalid address	The address value is invalid. Specify a correct one.
0503	User break	Processing was interrupted by the break key input.
0505	Not found	Not found.
0507	Cannot load file	Loading failed.
0508	Cannot save file	Saving failed.
0509	Divide by zero	The divisor in the integer expression is 0. Change it to a value other than 0.
0510	Number out of range	Data outside the range was specified.
0511	Invalid command	An invalid command was executed.
0512	Invalid operator	The operator is invalid.
0513	Mismatched parentheses	The parentheses are not matched.
0514	Invalid character constant	The character constant is invalid.
0515	Invalid register name	The register name is invalid.
0516	Invalid function name	The function name is invalid.
0517	File write error	A file write error occurred.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
0519	Out of analysis space	The analysis range was exceeded.
0520	Analysis ranges overlap	The analysis ranges overlapped.
0521	Not an analysis range	It is not an analysis range.
0522	No trace data available	No valid trace data was available.
0525	File verify error	A difference was found during file verification.
0526	File format error	The file format is invalid.
0532	Cannot load as program. No Source level debugging available	The user program has not been loaded. It is assumed that debug option was not specified when creating the program. Source level debugging cannot be performed.
0533	File does not exist	The specified file does not exist.
0534	Not enough memory	Memory to be used by the simulator/debugger cannot be allocated. Extend memory or change the user program.
0536	No function selected	No function has been selected.
0537	File read error	A file read error occurred.
0600	Not logging	Logging by a LOG command has not been executed.
0601	No log file set	No LOG file has been specified.
0602	Program did not start.	The user program has not been executed.
0603	Program has stopped.	Execution of the user program has stopped.
0604	Must specify go till address.	Specify the break address.
0606	Memory map not available	No memory map has been set.
0607	Trace record out of range	The traced range was exceeded.
0609	Trace not available	Trace is invalid.
0905	Invalid expression	An invalid expression was used.
0906	String too long	The character string is too long.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
1002	Register not found	The register was not found.
1003	Invalid register index	The register index is invalid.
1006	Invalid memory map mode specified	An invalid memory map mode was specified.
1007	Invalid endian type	An invalid endian type was specified.
1500	Invalid mnemonic	The mnemonic is invalid.
1501	Invalid operand	The operand is invalid.
1502	Syntax error	The command parameter is invalid.
1503	Too many operands	Too many operands were specified.
1504	Operand out of range	The address specified by the operand exceeded the range.
1505	Bad address operand alignment	The address alignment is invalid.
1507	Invalid numeric constant	The input value is invalid.
1508	Divide by zero	The divisor in the integer expression was 0. Change it to a value other than 0.
1509	Invalid mnemonic specifier	An illegal mnemonic was described.
2001	Not currently available	The entered command is not supported by the simulator/debugger or cannot be executed now, or the reset vector is not set to an readable memory area. Set the reset vector to a readable/writable memory area.
2010	Invalid address value	The address value is invalid.
2011	Invalid length value	An invalid length was specified.
2012	Invalid index value	An invalid index was specified.
2013	Invalid memory space value	An invalid memory map was set.
2019	Invalid parameter	An invalid parameter was specified.
2020	Compare failed	Non-conformance was found during comparison.
2021	Find failed	No conforming item was found.
2022	Verify failed	Non-conformance was found during verification.
2023	Table full	The table has become full.
2030	Not an instruction	The specified breakpoint address was not an instruction.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
2100	Invalid address	An invalid address was specified.
2101	Invalid count	An invalid count was specified.
2102	Too many parameter	Too many parameters were specified.
2103	No breakpoint set	No breakpoint has been set.
2104	Invalid data	Invalid data was specified.
2105	Invalid size	An invalid size was specified.
2106	Invalid option	An invalid option was specified.
2107	No breakdata set	No break data has been set.
2108	Invalid start address	An invalid start address was specified.
2109	Invalid end address	An invalid end address was specified.
2110	Invalid access type	An invalid access type was specified.
2111	No breakaccess set	No break access has been set.
2112	Invalid register name	An invalid register name was specified.
2113	No set parameter	No parameter has been specified.
2114	No breaksequence set	No break sequence has been set.
2115	Invalid index value	An invalid index was specified.
2116	Get break failure	Break setting failed.
2117	No resource	No program has been loaded.
2118	No set performance range	No time analysis function has been set.
2119	No breakregister set	No break register has been set.
2120	Address re-use	The address was re-used.
2121	Can't read	Reading failed.
2122	Exception handling error	An exception handling error occurred.
2123	No get memory area	Memory map setting failed.
2124	Invalid address	An invalid address was specified.
2125	Address already use	The specified address has already been used.
2126	Exception error	An exception processing error occurred. Correct the user program to prevent the error from occurring.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
2127	Memory access error	One of the following states occurred: (1) A memory area that had not been allocated was accessed. (2) Data was written to a memory area having the write protect attribute. (3) Data was read from a memory area having the read disable attribute. (4) A memory area in which memory does not exist was accessed. Allocate memory, change the memory attribute, or correct the user program to prevent the memory from being accessed.
2128	Address error	One of the following states occurred: (1) A PC value was an odd number. (2) An instruction was read from the internal I/O area. (3) Word data was accessed to an address other than 2n. (4) Long-word data was accessed to an address other than 4n. (5) The VBR or SP was a value other than a multiple of 4. (6) An error occurred in the exception processing of an address error. Correct the user program to prevent the error from occurring.
2129	Memory already set	The memory map has already been set.
2130	Memory area not exist	The specified address has not been set for memory map.
2131	Invalid value	An invalid value was specified.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
2132	General invalid instruction	One of the following conditions caused a general invalid instruction error. (1) The program attempted to execute a code that is not an instruction. (2) An error occurred during exception processing of general invalid instructions. Correct the user program so that the error does not occur.
2133	Illegal operation	The following condition caused an illegal operation. (1) A zero division occurred during the DIV1 instruction. Correct the user program so that the error does not occur.
2134	Invalid slot instruction	One of the following conditions caused an invalid slot instruction error. (1) The branch instruction that changes PC immediately after the delayed branch instruction was executed. (2) An error occurred during exception processing of the invalid slot instruction. Correct the user program so that the error does not occur.
2135	Illegal DSP operation	Either of the following states occurred: (1) A shift exceeding 32 bits was attempted with the PSHA instruction. (2) A shift exceeding 16 bits was attempted with the PSHL instruction. Correct the user program to prevent the error from occurring.
2136	Invalid DSP instruction code	An invalid instruction code was detected in the DSP parallel instruction. Correct the user program to prevent the error from occurring.
2137	TLB miss	TLB miss occurred during simulation or command execution. Take necessary measures such as updating TLB contents.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
2138	TLB invalid	TLB invalid exception processing occurred during simulation or execution. Take necessary measures such as updating TLB contents.
2139	TLB protection violation	TLB protection violation occurred during simulation.
2140	Initial page write	Initial page writing occurred during simulation.
2141	TLB multiple hit	The virtual address access hit multiple TLB entries during simulation or execution. TLB is not set appropriately. Modify TLB contents and program (handler routine).
2142	Multiple exception	One of the following states occurred during floating-point operation: (1) An invalid operation (2) A division by zero Correct the user program to prevent the error from occurring.
2143	Illegal LRU set	The LRU value of the cache is illegal. Check the setting.
2144	Simulated I/O	A system call error occurred. Correct incorrect contents of registers R0, R1, and parameter block.
2145	System call error	A system call error occurred.
2146	FPU Error	One of the following states occurred during floating-point operation: (1) An FPU error occurred. (2) An invalid operation occurred. (3) A division by zero occurred. (4) An overflow occurred. (5) An underflow occurred. (6) An inaccurate operation occurred. Correct the user program to prevent the error from occurring.
2147	No get stack data	No stack information has been obtained.
2148	Unified TLB Miss	A unified TLB miss occurred during memory access. Take necessary procedures such as updating the unified TLB contents.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
2150	Unified TLB Protection Violation	A unified TLB protection exception occurred during memory access. Take necessary procedures such as updating the unified TLB contents.
2151	Initial Page Write	The initial page has been written to. Take necessary procedures such as updating the TLB contents.
2152	Unified TLB Multiple Hit	Multiple unified TLB entries were hit when a virtual address was accessed in memory. Unified TLB is not correctly set. Modify unified TLB contents and user program (handler routine).
2153	Instruction TLB Miss	An instruction TLB miss occurred during memory access. Take necessary procedures such as updating the instruction TLB contents.
2155	Instruction TLB Protection Violation	An instruction TLB protection exception occurred during memory access. Take necessary procedures such as updating the instruction TLB contents.
2156	Instruction TLB Multiple Hit	Multiple instruction TLB entries were hit when a virtual address was accessed in memory. Instruction TLB is not correctly set. Modify instruction TLB contents and user program (handler routine).
2157	FPU Disable	An attempt was made to execute an FPU instruction while the FPU is disabled (SR.FD = 1). Correct the user program to prevent the error from occurring.
2158	Slot FPU Disable	An attempt was made to execute an FPU instruction in a delay slot while the FPU is disabled (SR.FD = 1). Correct the user program so that no error occurs.
2159	Instruction TLB Illegal LRU	An LRU value in the instruction TLB is illegal. Check the setting.
2160	Illegal PR bit	An attempt was made to execute an FPU instruction while the PR bit value of the FPSCR is illegal. Correct the user program to prevent the error from occurring.
2161	Illegal Combination BSC Register	An attempt was made to access the area for which the BSC register setting is invalid. Correct the user program to prevent the error from occurring.

Table 5.2 Error Messages (cont)

No.	Message	Contents/Measures
3002	Incorrect object module format	The file format is incorrect.
3003	Object module allocation	Memory to be used by the simulator/debugger cannot be allocated. Extend memory or change the user program.
3004	Object module not absolute format	The file format is incorrect.
3005	Incorrect object module cpu	A user program for another CPU was attempted to be loaded.
3030	Can't convert	Conversion failed.
3041	Not enough memory	Memory is insufficient.

Section 6 Windows and Dialog Boxes

The table below lists the menu bars of the simulator/debugger and the corresponding pull-down menus.

Table 6.1 Menu and Function

Menu bar	Pull-down menu	Sub-menu	Function
File	Auto Typing		Inputs debugger commands from a file
	Logging		Outputs a file showing the debugger command results
	Recording		Saves the simulator/debugger operation procedure
	Replaying		Replays the saved operation procedure
	File Selection		Selects the file name
	File Load		Load
	Load Status		Restores the debugger status
	File Save		Saves
	Save Status		Saves the debugger status
	Quit		Stops the simulator/debugger
View	Register		Displays and changes registers
	Disassemble		Displays disassembled program
	Show Calls		Displays function calls
	Source Files		Displays source files and their names
	Function List		Displays function names and source files
	Expression Value		Displays expression values
	Localized Dump		Localizes the memory contents to Japanese
	Stack		Displays the stack contents
	Symbol List	Symbol	Displays variable names whose contents can be displayed at the PC address
		Object	Displays the object names of the current scope
		Class	Displays the class names of the current scope
	Symbol Value	Symbol Value No. x	Displays and changes the variable contents (x = 1 to 4)

Table 6.1 Menu and Function (cont)

Menu bar	Pull-down menu	Sub-menu	Function
View	Dump	Dump No. x	Displays the memory contents (x = 1 to 4)
	Analysis	Performance	Sets and displays execution performance analysis
		Stack	Sets and displays stack trace
Execute	Go		Starts execution
	Exec Mode		Sets and displays execution mode
Break	Break		Sets, displays, and clears breakpoints
	Break Access		Sets, displays, and clears the break condition based on access
	Break Data		Sets, displays and clears the break condition based on memory value
	Break Register		Sets, displays and clears the break condition based on register value
	Break Sequence		Sets, displays and clears the break condition based on execution sequence
Trace	Trace		Displays trace information
	Trace Condition		Sets the trace condition and starts and stops trace
Help	General Operation		Explains simulator/debugger operation procedure
	Symbolic Input		Explains address symbolic specifications
	Command Name		Explains debugger commands

Section 7 How to Create CPU Information File

7.1 Functions of CPU Information File Creating Program (CIA)

The simulator/debugger uses a CPU information file to load sections of each microcomputer according to memory map and check that sections have not been loaded crossing memory type boundaries. A CPU information file is created using the CIA (CPU Information Analyzer).

The CIA has the following three functions.

(1) CPU information file creation
Creates the CPU memory map information file of the microcomputer to be used.
(2) CPU information file display
Allows the contents of the generated CPU information file to be checked.
(3) CPU information editing (deletion/addition)
Allows the contents of the generated CPU information file to be modified by deleting or adding.

7.2 Invoking CIA

The format of the command line used to invoke the CIA program is shown below.

- SH-1, SH-2, SH-3, SH-2E, and SH-3E
  % ***ciashΔ<CPU information file name>(RET)***
- SH-DSP
  % ***ciashdspΔ<CPU information file name>(RET)***
- SH-4
  % ***ciash4Δ<CPU information file name>(RET)***

Either an existent or a new CPU information file can be specified. When an existent CPU information file is specified, the program requests the input of a name for the output CPU information file. If the file extension is omitted, the extension .cpu is supplied as default.

7.3 CIA Usage Procedures and Selection Menus

Figure 7.1 shows the procedure used with the CIA program.

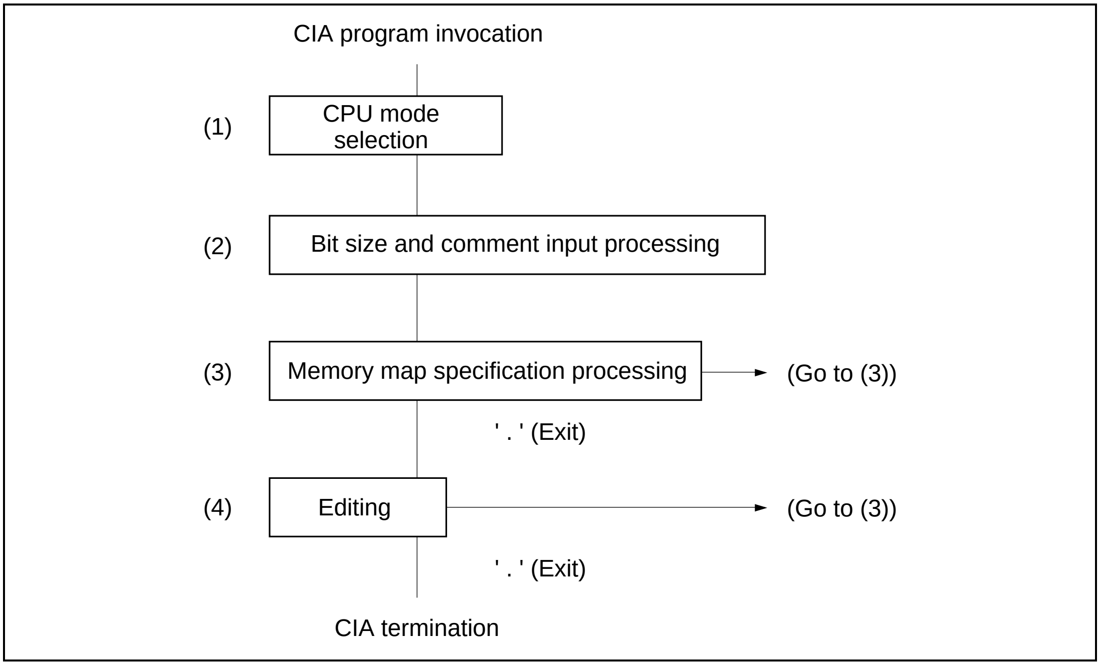


Figure 7.1 CIA Usage Procedure

(1) Mode selection

— ciash

A CPU is selected among SH-1, SH-2, SH-3 and SH-3E.

— ciashdsp

Either internal exception vector mode or external exception vector memory mode is selected.

— ciash4

No mode is selected.

(2) Comment input

A comment can be specified to identify the CPU information. Up to 127 characters can be specified.

The comment can only be input when creating a new CPU information file. The CIA procedure starts with step (4), Editing, when an existent CPU information file is specified.

(3) Memory map specification

The ROM, EXTERNAL, RAM, or IO can be selected as memory map specification. Memory map specification is iterated until a period (the exit command) is specified.

(4) Editing

The following options are presented as a CPU information editing menu.

— ciash, ciashdsp

1:ADD 2:DELETE 3:COMMENT 4:CIA ABORT .:CIA END

- When '1'(ADD) is selected, the memory map specification of step 3 is performed.
- When '2'(DELETE) is selected, the system prompts (by index number) for input of an address range to be deleted.
- When '3'(COMMENT) is selected, the system prompts for input of a new comment line.
- When '4'(CIA ABORT) is selected, CIA processing is terminated without saving the CPU information file.
- When '.'(CIA END) is selected, the system writes the memory map information to the CPU information file and completes CIA processing normally.

— ciash4
1:MODIFY 2:MODIFY 3:COMMENT 4:CIA ABORT .:CIA END

- When '1'(MODIFY) is selected, the memory map specification of step 3 is performed.
- When '2'(MODIFY) is selected, the memory map specification of step 3 is performed (same as item 1).
- When '3'(COMMENT) is selected, the system prompts for input of a new comment line.
- When '4'(CIA ABORT) is selected, CIA processing is terminated without saving the CPU information file.
- When '.'(CIA END) is selected, the system writes the memory map information to the CPU information file and completes CIA processing normally.

7.4 CIA Sample Sessions

This section explains typical CIA use when the SH-3 is used. The bold italic face is the input by the user.

(1) Creating a new CPU information file

```
% ciash sh3                                                                ---(1)

SH SERIES CIA Ver. 3.0 (HS0700CICS3SM)
Copyright (C) Hitachi, Ltd. 1992, 1996
Licensed Material of Hitachi, Ltd.

*** NEW FILE ***

    *** CPU MENU ***
    1:SH1  2:SH2  3:SH3  4:SH3E
    ? 3                                                                    ---(2)
  BIT SIZE 32 ? :32                                                        ---(3)
  COMMENT?      :SH3 CPU INFORMATION                                       ---(4)

                 *** MAP MENU ***
                 0:ROM  1:EXTERNAL  2:RAM  3:I/O  .:END
                 ? 1                                                       ---(5)
                      * EXTERNAL START ADDRESS?  0                         ---(6)
                                 END   ADDRESS?  fffff                     ---(7)
                        STATE COUNT           ?  3                         ---(8)
                        DATA BUS SIZE         ?  32                        ---(9)
                      * EXTERNAL START ADDRESS?  . (period input)          ---(10)

                 *** MAP MENU ***
                 0:ROM  1:EXTERNAL  2:RAM  3:I/O  .:END
                 ? . (period input)

                       ***** CPU INFORMATION *****
                       CPU : SH3                                         ---(11) (a)
                       SH3 CPU INFORMATION                               ---(11) (b)
                       BIT SIZE : 32                                     ---(11) (c)
                          No Device   Start    End      State Bus
                           1:EXTERNAL:00000000-000FFFFF  3    32         ---(11) (d)-(i)
```

```
        *** EDIT MENU ***
        1:ADD  2:DELETE  3:COMMENT  4:CIA ABORT  .:CIA END
        ? . (period input)                                              ---(12)

*** CIA COMPLETED ***
%
```

Description:

(1) Specify the name of a new CPU information file when the CIA program is invoked.

(2) Specify the CPU type.

(3) Specify the bit size in decimal. The displayed default is taken if the specification is omitted.

(4) Specify a comment. The comment field is left blank if this line is omitted. If more than 127 characters are entered, a warning message is displayed and the characters following the first 127 are ignored.

(5) Specify the memory type as a number corresponding to the input menu.

(6) Specify the start address of the corresponding memory area in hexadecimal.

(7) Specify the end address of the corresponding memory area in hexadecimal.

(8) Specify the number of states for the corresponding memory area in decimal.

(9) Specify the data bus width for the corresponding memory area in decimal.

(10) Terminate data entry for the corresponding memory area with a period ('.').

(11) The edit menu is automatically displayed when the input menu is terminated.

   (a) The CPU type

   (b) The comment

   (c) The bit size

   (d) The map number

   (e) The memory type

   (f) The start address

   (g) The end address

   (h) The number of states

   (i) The data bus width

(12) Inputting a period terminates CIA processing normally. The memory map data is written to the file (sh3.cpu) specified when the CIA program was invoked.

(2) Editing a CPU information file

The address range and data bus width of the external memory are changed as follows:

```
% ciash sh1.cpu                                                                  ---(1)

SH SERIES CIA Ver. 3.0 (HS0700CICS3SM)
Copyright (C) Hitachi, Ltd. 1992, 1996
Licensed Material of Hitachi, Ltd.

*** OLD FILE ***
  NEW CPU FILE NAME? sh1.cpu                                                     ---(2)

                    ***** CPU INFORMATION *****
                    CPU : SH1
                    SH1 CPU INFORMATION
                    BIT SIZE : 28
                      No   Device     Start       End      State  Bus
                       1 : ROM AREA : 00000000 - 0000FFFF  1     32
                       2 : EXTERNAL : 01000000 - 010FFFFF  4     16
                       3 : RAM AREA : 0F000000 - 0F000FFF  1     32

      *** EDIT MENU ***
      1:ADD  2:DELETE  3:COMMENT  4:CIA ABORT  .:CIA END
      ? 2                                                                        ---(3)
      DELETE MAP NUMBER? 2                                                       ---(4)


                    ***** CPU INFORMATION *****
                    CPU : SH1
                    SH1 CPU INFORMATION
                    BIT SIZE : 28
                      No   Device    Start       End       State  Bus
                       1 : ROM AREA : 00000000 - 0000FFFF  1     32
                       2 : RAM AREA : 0F000000 - 0F000FFF  1     32
      *** EDIT MENU ***
      1:ADD  2:DELETE  3:COMMENT  4:CIA ABORT  .:CIA END
      ? 1                                                              ---(5)

           *** MAP MENU ***
           0:ROM  1:EXTERNAL  2:RAM  3:I/O  .:END
           ? 1                                                         ---(6)
                * EXTERNAL START ADDRESS?  9000000
                           END   ADDRESS?  90fffff
                  STATE COUNT          ?  4
                  DATA BUS SIZE        ?  8
                * EXTERNAL START ADDRESS?  . (period input)

           *** MAP MENU ***
           0:ROM  1:EXTERNAL  2:RAM  3:I/O  .:END
           ? . (period input)                                          ---(7)

                    ***** CPU INFORMATION *****
                    CPU : SH1
                    SH1 CPU INFORMATION
                    BIT SIZE : 28
                      No   Device     Start      End      State  Bus
                       1 : ROM AREA : 00000000 - 0000FFFF  1     32
                       2 : EXTERNAL : 09000000 - 090FFFFF  4      8
                       3 : RAM AREA : 0F000000 - 0F000FFF  1     32

      *** EDIT MENU ***
      1:ADD  2:DELETE  3:COMMENT  4:CIA ABORT  .:CIA END
      ? . (period input)

*** CIA COMPLETED ***
%
```

Description

(1) Specify the name of the file to be edited when the CIA program is invoked.
The extension .cpu is supplied if the file extension is omitted.

(2) Specify a new file to be created when editing is completed. If only (RET) is entered, the data will be output to the file specified in item (1). If only the file extension is omitted, the extension .cpu will be supplied. The map data is automatically displayed.

(3) Specify DELETE to delete information to be changed in the edit menu.

(4) Specify the information to be deleted as a map number. The state of the map information after the deletion is displayed.

(5) Specify ADD to input the changed information.

(6) The input menu is displayed, and the memory type is entered in the same manner as that used when creating a new CPU information file. The state of the map information after the addition is displayed.

(7) If END is specified, map information of added results will be displayed.

7.5 CIA Limitations

Table 7.1 lists the limitations on data specified using the CIA program. The CIA program cannot handle values which exceed these limitations.

Table 7.1 CIA Limitations

Item	Limitation Value
Input file format	• CPU information files output by the CIA
Bit size	• Only values specified in decimal • The specifiable range is from 24 to 32
Address specifications	• Only values specified in hexadecimal • The specifiable range depends on the bit size
Memory area	• Only values specified in decimal • The specifiable range is from 1 to 64
Number of states	• Only values specified in decimal • The specifiable range is from 1 to 65535
Data bus width	• Only values specified in decimal • The specifiable values are multiples of 8 between 8 and 65528
Comment length	• Up to 127 characters
Number of map information items	• Up to 65535 items

SuperH^{TM} RISC engine Simulator/Debugger User's Manual

Publication Date: 1st Edition, September 1999

Published by:	Electronic Devices Sales & Marketing Group Semiconductor & Integrated Circuits Hitachi, Ltd.
Edited by:	Technical Documentation Group UL Media Co., Ltd.