To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.

Hitachi Microcomputer Development Environment System

H8S, H8/300 Series Simulator/Debugger

User's Manual

ADE-702-282A

Rev. 2.0
05/14/02
Hitachi, Ltd.

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

Trademarks:

Microsoft® and Windows® are registered trademarks of Microsoft Corporation in the United States and/or other countries.

IBM PC is the name of a computer administered by International Business Machines Corporation.

ELF/DWARF2 is the name of an object format developed by the Tool Interface Standards Committee.

All products or brand names used in the manual are trademarks or registered trademarks of their respective companies.

Read First:

1. Hitachi, Ltd. (including its subsidiaries, hereafter collectively referred to as Hitachi) pursues a policy of continuing improvement in design, performance, and safety of the system. Hitachi reserves the right to change, wholly or partially, the specifications, design, user's manual, and other documentation at any time without notice.
2. This user's manual and this system are copyrighted and all rights are reserved by Hitachi. No part of this user's manual, all or part, may be reproduced or duplicated in any form, in hard-copy or machine-readable form, by any means available without Hitachi's prior written consent.
3. Hitachi assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.

Preface

Read First

READ this user's manual before using the simulator debugger.

KEEP the user's manual handy for future reference.

Do not attempt to use the system until you fully understand its mechanism.

About this Manual

This manual explains the use of the simulator debugger and the Hitachi Embedded Workshop (HEW) for Hitachi microcomputer development tools. The following section will provide a brief *Introduction* to the debugging interface and simulator/debugger, and list its key features.

The following sections, *System Overview*, *Simulator/Debugger Functions*, *Menus*, *Windows and Dialog Boxes*, *Command Lines*, and *Messages*, give reference information about the operation and facilities available from these respective areas.

This manual assumes that the HEW is used on the English version of Microsoft® Windows®Me operating system running on the IBM PC.

Assumptions

It is assumed that the reader has a competent knowledge of the C/C++ programming language, assembly-language mnemonics for the processor being debugged and is experienced in using Microsoft® Windows® applications.

Document Conventions

This manual uses the following typographic conventions:

Table 1 Typographic Conventions

CONVENTION	MEANING
[Menu->Menu Option]	Bold text with '->' is used to indicate menu options (for example, **[File->Save As...]**).
FILENAME.C	Uppercase names are used to indicate file names.
"enter this string"	Used to indicate text that must be entered (excluding the " " quotes).
Key+Key	Used to indicate required key presses. For example, **Ctrl+N** means press the Ctrl key and then, while holding the Ctrl key down, press the N key.
➲ (The "how to" symbol)	When this symbol is used, it is always located in the left-hand margin. It indicates that the text to its immediate right is describing "how to" do something.

Contents

Figures

Tables

Section 1 Overview

The Hitachi Embedded Workshop (HEW) is a Graphical User Interface intended to ease the development and debugging of applications written in C/C++ programming language and assembly language for Hitachi microcomputers. Its aim is to provide a powerful yet intuitive way of accessing, observing and modifying the debugging platform in which the application is running.

Key Features

- **Intuitive interface**
- **On-line help**
- **Common "Look & Feel"**

Note: The HEW does not run on Windows® version 3.1.

The simulator/debugger provides simulation functions for H8S, H8/300 series microprocessors and provides debugging functions for programs written in C, C++, or assembly language. Therefore, the simulator/debugger promotes efficient debugging of programs.

When used with the following software, the simulator/debugger reduces the time required for software development.

- Hitachi Embedded Workshop (HEW)
- H8S, H8/300 series C/C++ compiler
- H8S, H8/300 series cross assembler
- Optimizing linkage editor

1.1 Features

- Since the simulator/debugger runs on a host computer, software debugging can start without using an actual user system, thus reducing overall system development time.
- The simulator/debugger calculates the number of instruction execution cycles for a program during simulation, thus enabling performance evaluation without using an actual user system.
- The simulator/debugger offers the following features and functions that enable efficient program testing and debugging.
  - The ability to handle all of the H8S, H8/300 series CPUs
  - Functions to stop or continue execution when an error occurs during user program execution
  - Profile data acquisition and function-unit performance measurement
  - A comprehensive set of break functions (Pseudo interrupts are also possible)
  - Functions to set or edit memory maps
  - Functions to display function call history
  - Coverage information is displayed at the C/C++ or assembly-source level
  - Visual debugging functions provide the display of data as images or waveforms
- The breakpoint, memory map, performance, and trace can be set through the dialog box under Windows®. Environments corresponding to each memory map of the H8S, H8/300 microprocessors can be set through the dialog box.

1.2 Target User Program

Load modules in ELF/DWARF2 format and S-type format can be debugged with the simulator/debugger. These load modules are called user programs in this manual.

Figure 1.1 shows the creation of target user programs to be debugged.

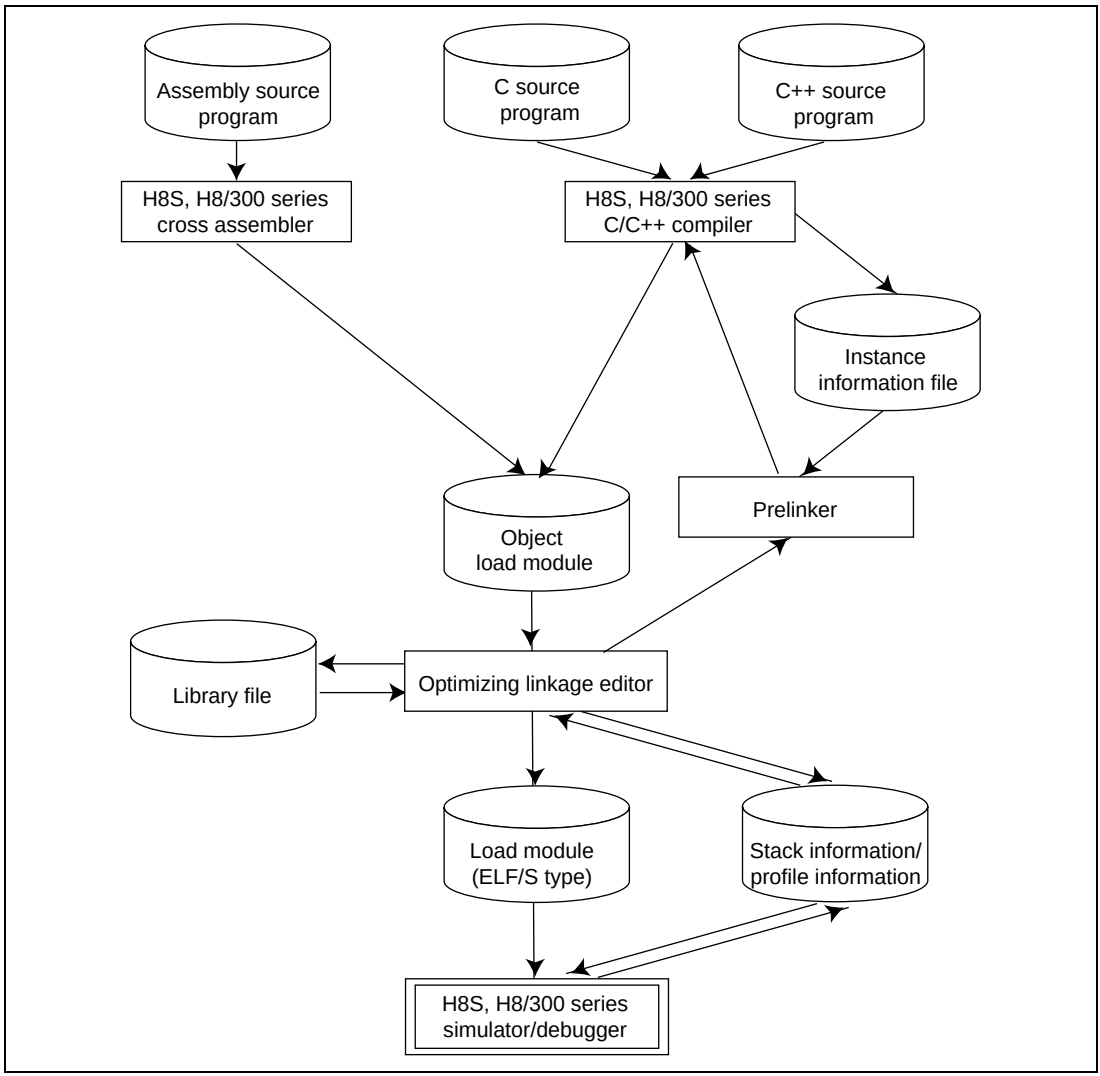


Figure 1.1 Creation of Target User Programs

1.3 Simulation Range

Simulation of the H8/300, H8/300L, H8S/2600, and H8S/2000 series microcomputers is supported. The simulator/debugger supports the following H8S, H8/300 series microcomputer functions:

- All CPU instructions
- Exception processing
- Registers
- All address areas
- CPU modes shown in table 1.1

Table 1.1 Platforms and the Corresponding CPU Modes

Names of Debugging Platforms	Corresponding CPU Modes
H8/300 Simulator	H8/300
H8/300L Simulator	H8/300L
H8/300HA Simulator	H8/300H Advanced Mode
H8/300HN Simulator	H8/300H Normal Mode
H8S/2600A Simulator	H8S/2600 Advanced Mode
H8S/2600N Simulator	H8S/2600 Normal Mode
H8S/2000A Simulator	H8S/2000 Advanced Mode
H8S/2000N Simulator	H8S/2000 Normal Mode

The simulator/debugger does not support the following H8S, H8/300 series MCU functions. Programs that use these functions must be debugged with the H8S, H8/300 series emulator.

- Dual-port RAM
- Timer
- Pulse-width modulator (PWM)
- Serial communication interface (SCI)
- A/D converter
- I/O ports
- Interrupt controller

Section 2 System Overview

HEW is a modular software system, utilizing self-contained modules for specific tasks. These modules are linked to a general purpose Graphical User Interface, which provides a *common look & feel* independent of the particular modules with which the system is configured.

2.1 User Interface

The HEW Graphical User Interface is a Windows® application that presents the debugging platform to you and allows you to set up and modify the system. Refer to a standard Windows® user manual for details on how to operate within a Windows® application.

2.2 Data Entry

When entering numbers in any dialog box or field you can always enter an expression instead of a simple number. This expression can contain symbols and can use the operators in the C/C++ programming languages. Use of C/C++ programming language features such as arrays and structures is only available if the ELF/DWARF2 format that supports C/C++ programming language debugging is in use.

In some dialogs, where there is a control expecting an end address, it is possible to enter a range by prefixing the value with a + sign. This will set the actual end address to be equal to the start address plus the entered the value.

2.2.1 Operators

The C/C++ programming language operators are available:

+, -, *, /, &, |, ^, ~, !, >>, <<, %, (,), <, >, <=, >=, ==, !=, &&, ||

2.2.2 Data Formats

Unprefixed data values will be taken as being in the default radix set by the **[Options->Radix]** menu option. The exception is count field which use decimal values by default (independent of the current default system radix).

Symbols may be used by name and ASCII character strings can be entered if surrounded by single quote characters, e.g. ‘demo’.

The following prefixes can be used to identify radices:

B’	Binary
O’	Octal
D’	Decimal
H’	Hexadecimal
0x	Hexadecimal

The contents of a register may be used by specifying the register name, prefixed by the # character, e.g.:

#R1, #E1

2.2.3 Precision

All mathematics in expression evaluation is done using 64 bits (signed). Any values exceeding 64 bits are truncated.

2.2.4 Expression Examples

```
Buffer_start + 0x1000
#R1 | B’10001101
((pointer + (2 * increment_size)) & H’FFFF0000) >> D’15
!(flag ^ #R4)
```

2.2.5 Symbol Format

You can specify and reference symbols in the same format as in C/C++ programming language. Cast operators may be used together with symbols, and you can reference data after its type has been converted. Note the following limitations.

- Pointers can be specified up to four levels.
- Arrays can be specified up to three dimensions.
- No typedef name can be used.

2.2.6 Symbol Examples

`Object.value`	: Specifies direct reference of a member (C/C++)
`p_Object->value`	: Specifies indirect reference of a member (C/C++)
`Class::value`	: Specifies reference of a member with class (C++)
`*value`	: Specifies a pointer (C/C++)
`array[0]`	: Specifies an array (C/C++)
`Object.*value`	: Specifies reference of a pointer to member (C++)
`::g_value`	: Specifies reference of a global variable (C/C++)
`Class::function(short)`	: Specifies a member function (C++)
`(struct STR) *value`	: Specifies cast operation (C/C++)

Section 3 Simulator/Debugger Functions

This section describes the functions of the H8, H8/300 series simulator/debugger.

3.1 Simulator/Debugger Memory Management

The setting for the memory map is used to calculate the number of memory access cycles during simulation. The simulator/debugger supports the memory types listed in table 3.1.

Table 3.1 Memory Types

Memory Types	Execution of Programs for Debugging
Internal ROM	Enabled
Internal RAM	Enabled
External Memory	Enabled
Internal I/O	Disabled
EEPROM	Enabled

3.1.1 Memory Map Specification

A memory map can be specified in the **Simulator System** dialog box to calculate the number of memory access cycles during simulation.

The following items can be specified:

- Memory type
- Start and end addresses of the memory area
- Number of memory access cycles
- Memory data bus width

The memory types that can be specified depend on the CPU. For details, refer to section 5.24, Simulation System Dialog Box. The user program can be executed in all areas except for the internal I/O area.

3.1.2 Memory Resource Specification

A memory resource must be specified to load and execute a user program.

The memory resource, including the following items, can be specified in the **Simulator Memory Resource** dialog box.

- Start address
- End address
- Access type

The access type can be read/write, read-only, or write-only. Since an error occurs if the user program attempts an illegal access (for example, trying to write to a read-only memory), such an illegal access in the user program can be easily detected. Unlike other types of memory, an EEPROM can be written to by an EEPMOV instruction even when the access type is read-only. If writing is allowed, the EEPROM cannot be written to by instructions other than EEPMOV.

3.2 Instruction Execution Reset Processing

The simulator/debugger resets the number of instruction executions and number of cycles that instructions are executed when:

- The program counter (PC) is modified after the instruction simulation stops and before it restarts.
- The Run command to which the execution start address has been specified is executed.
- Initialization is performed, or a program is loaded.

3.3 Exception Processing

The simulator/debugger detects the generation of exceptions corresponding to TRAPA instructions (only for the H8/300H and H8S series) and trace exceptions (only for the H8S series). In addition, the simulator/debugger simulates exception processing.

The simulator/debugger simulates exception processing with the following procedures.

1. Detects an exception during instruction execution.
2. Saves the PC and CCR in the stack area. Also saves EXR when the valid bit of EXR is on. When an error occurs, simulator/debugger stops exception processing, displays that an exception processing error has occurred, and enters the command input wait state.
3. Sets bit 1 in the CCR to 1.
4. Reads the start address from the vector address corresponding to the vector number. When an error occurs, simulator/debugger stops exception processing, displays that an exception processing error has occurred, and enters the command input wait state.
5. 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.

3.4 Features Specific to the H8S/2600 CPU

The H8S/2600 CPU offers the following specific features.

- Execution of a multiply and accumulate (MAC) instruction

  Either saturation or non-saturation multiply and accumulate operation can be executed depending on the state of bit 7 (the MACS bit) in the SYSCR register in the internal I/O area:

  MACS bit = 0: Non-saturation operation
  MACS bit = 1: Saturation operation
- Enabling or disabling the EXR register

  The EXR register can be enabled or disabled depending on the state of bit 5 (the EXR bit) in the SYSCR register in the internal I/O area:

  EXR bit = 0: EXR disabled
  EXR bit = 1: EXR enabled

The SYSCR register address can be specified using **[SYSCR Address]** in the **Simulator System** dialog box.

Note: The SYSCR register address must be within the internal I/O area; otherwise the simulator/debugger assumes the MACS bit as 0 (non-saturation operation) and EXR bit as 0 (EXR disabled).

For details, refer to section 5.24, Simulator System Dialog Box.

3.5 Control Registers

The simulator/debugger supports SYSCR (system control register) as a memory-mapped control register for the H8S/2600 CPU. Therefore, a user program using a multiply and accumulate operation and EXR access can be simulated and debugged.

The SYSCR address can be specified in **[SYSCR Address]** of the **Simulator System** dialog box.

To modify or display a control register value, use the **IO** window. For details, refer to section 5.24, Simulator System Dialog Box and section 5.6, IO.

3.6 Trace

The simulator/debugger writes the results of each instruction execution into the trace buffer. The conditions for the trace information acquisition can be specified in the **Trace Acquisition** dialog box. Click the right mouse button in the **Trace** window and choose **[Acquisition...]** from the popup menu to display the **Trace Acquisition** dialog box. The acquired trace information is displayed in the **Trace** window. The trace information displayed in the **Trace** window depends on the target CPU as follows.

- Total number of instruction execution cycles
- Instruction address
- CCR
- Multiplier internal flag (valid only for the H8S/2600 series)
- Instruction mnemonic
- Data access information (destination and accessed data)
- C/C++ or assembly-language source programs

The trace information can be searched. The search conditions can be specified in the **Trace Search** dialog box. Click the right mouse button in the **Trace** window and choose **[Find...]** from the popup menu to display the **Trace Search** dialog box.

For details, refer to section 5.17, Trace Window, through section 5.19, Trace Search Dialog Box.

3.7 Standard I/O and File I/O Processing

The simulator/debugger provides the **Simulated I/O** window to enable the standard I/O and file I/O processing listed in table 3.6 to be executed by the user program. When the I/O processing is executed, the **Simulated I/O** window must be open. There are three types of function codes: a 16-bit address version, a 24-bit address version, and a 32-bit address version. Select the type according to the CPU to be used.

Table 3.2 I/O Functions

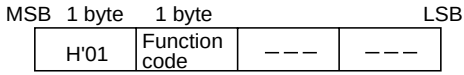
No.	Function Code	Function Name	Description
1	H’01 (16-bit address) H’11 (24-bit address) H’21 (32-bit address)	GETC	Inputs one byte from the standard input device
2	H’02 (16-bit address) H’12 (24-bit address) H’22 (32-bit address)	PUTC	Outputs one byte to the standard output device
3	H’03 (16-bit address) H’13 (24-bit address) H’23 (32-bit address)	GETS	Inputs one line from the standard input device
4	H’04 (16-bit address) H’14 (24-bit address) H’24 (32-bit address)	PUTS	Outputs one line to the standard output device
5	H’05 (16-bit address) H’15 (24-bit address) H’25 (32-bit address)	FOPEN	Opens a file
6	H'06	FCLOSE	Closes a file
7	H’07 (16-bit address) H’17 (24-bit address) H’27 (32-bit address)	FGETC	Inputs one byte from a file
8	H’08 (16-bit address) H’18 (24-bit address) H’28 (32-bit address)	FPUTC	Outputs one byte to a file
9	H’09 (16-bit address) H’19 (24-bit address) H’29 (32-bit address)	FGETS	Inputs one line from a file
10	H’0A (16-bit address) H’1A (24-bit address) H’2A (32-bit address)	FPUTS	Outputs one line to a file
11	H'0B	FEOF	Checks for end of file
12	H'0C	FSEEK	Moves the file pointer
13	H'0D	FTELL	Returns the current position of the file pointer

To perform I/O processing, use the **[System Call Address]** in the **Simulator System** dialog box in the following procedure.

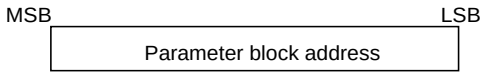
1. Set the address specialized for I/O processing in the **[System Call Address]**, select **[Enable]** and execute the program.
2. When detecting a subroutine call instruction (BSR or JSR), that is, a system call to the specialized address during user program execution, the simulator/debugger performs I/O processing by using the R0 and R1 (H8/300, H8/300L series) or ER1 (H8/300H, H8S series) values as the parameters.

Therefore, before issuing a system call, set as follows in the user program:

- Set the function code (table 3.2) to the R0 register



- Set the parameter block address to the R1 register (for the parameter block, refer to each function description)



- Reserve the parameter block and input/output buffer areas

Each parameter of the parameter block must be accessed in the parameter size.

After the I/O processing, the simulator/debugger resumes simulation from the instruction that follows the system call instruction.

Each I/O function is described in the following format:

(1)	(2)	(4)
	(3)	

Parameter Block

(5)

Parameters

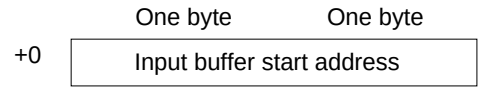
(6)

(1) Number corresponding to table 3.2
(2) Function name
(3) Function code
(4) I/O overview
(5) I/O parameter block
(6) I/O parameters

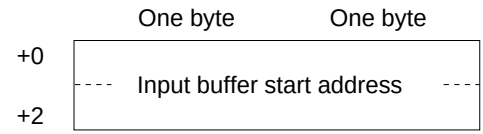
1	GETC	Inputs one byte from the standard input device
	H’01, H’11, and H'21	

Parameter Block

Function code: H'01 (16-bit address version)



Function code: H'11 (24-bit address version),
H'21 (32-bit address version)



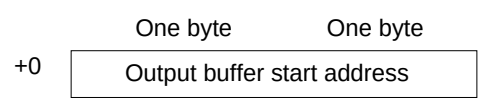
Parameters

- Input buffer start address (input)
  Start address of the buffer to which the input data is written.

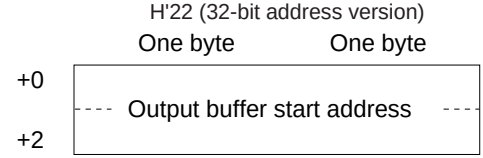
2	PUTC	Outputs one byte to the standard output device
	H’02, H’12, and H'22	

Parameter Block

Function code: H'02 (16-bit address version)



Function code: H'12 (24-bit address version),
H'22 (32-bit address version)



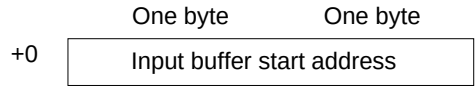
Parameters

- Output buffer start address (input)
  Start address of the buffer in which the output data is stored.

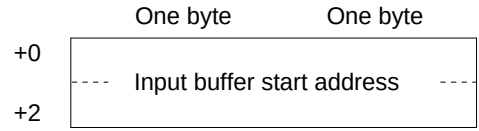
3	GETS H’03, H’13, and H'23	Inputs one line from the standard input device

Parameter Block

Function code: H'03 (16-bit address version)



Function code: H'13 (24-bit address version),
H'23 (32-bit address version)



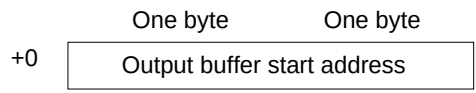
Parameters

- Input buffer start address (input)
  Start address of the buffer to which the input data is written.

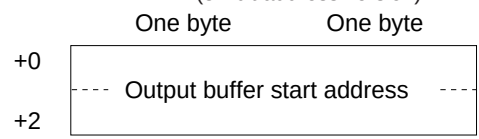
4	PUTS H’04, H’14, and H'24	Outputs one line to the standard output device

Parameter Block

Function code: H'04 (16-bit address version)



Function code: H'14 (24-bit address version),
H'24 (32-bit address version)



Parameters

- Output buffer start address (input)
  Start address of the buffer in which the output data is stored.

5	FOPEN	Opens a file
	H’05, H’15, and H'25	

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.

Parameter Block

Function code: H'05 (16-bit address version)

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

Function code: H'15 (24-bit address version),
H'25 (32-bit address version)

	One byte	One byte
+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 file accesses 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.

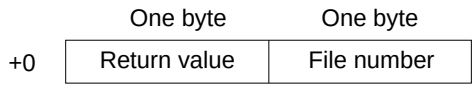
"a+" : Open for reading and appending.

"b" : Open in binary mode.

- Start address of file name (input)
  The start address of the area for storing the file name.

6	FCLOSE	Closes a file
	H'06	

Parameter Block



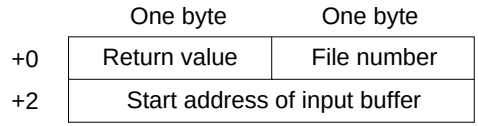
Parameters

- Return value (output)
  0: Normal completion
  –1: Error
- File number (input)
  The number returned when the file was opened.

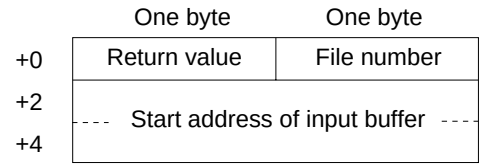
7	FGETC	Inputs one byte from a file
	H’07, H’17, and H'27	

Parameter Block

Function code: H'07 (16-bit address version)



Function code: H'17 (24-bit address version),
H'27 (32-bit address version)



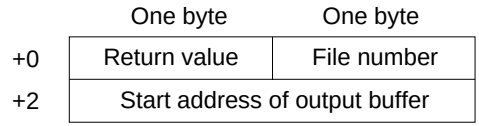
Parameters

- Return value (output)
  0: Normal completion
  –1: EOF detected
- 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.

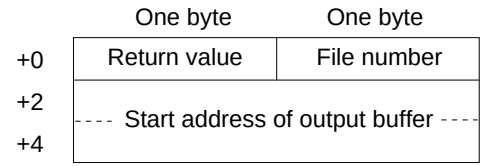
8	FPUTC	Outputs one byte to a file
	H’08, H’18, and H'28	

Parameter Block

Function code: H'08 (16-bit address version)



Function code: H'18 (24-bit address version),
H'28 (32-bit address version)



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	Reads character string data from a file
	H’09, H’19, and H'29	

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.

Parameter Block

Function code: H'09 (16-bit address version)

	One byte	One byte
+0	Return value	File number
+2	Start address of input buffer	

Function code: H'19 (24-bit address version),
H'29 (32-bit address version)

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

Parameters

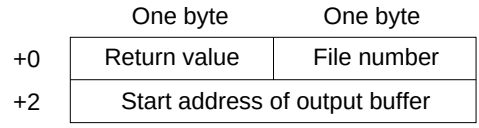
- Return value (output)
  0: Normal completion
  –1: EOF detected
- File number (input)
  The number returned when the file was opened.
- Buffer size (input)
  The size of the area for storing the read data. A maximum of 256 bytes can be stored.
- Start address of input buffer (input)
  The start address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

10	FPUTS	Writes character string data to a file
	H'0A H'1A, and H'2A	

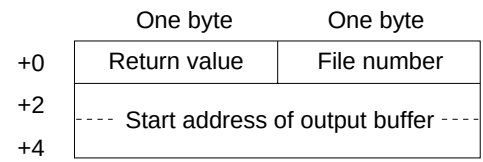
Writes character string data to a file. The NULL code that terminates the character string is not written to the file.

Parameter Block

Function code: H'0A (16-bit address version)



Function code: H'1A (24-bit address version),
H'2A (32-bit address version)



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.

11	FEOF	Checks for end of file
	H'0B	

Parameter Block

	One byte	One byte
+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	Moves the file pointer to the specified position
	H'0C	

Parameter Block

	One byte	One byte
+0	Return value	File number
+2	Direction	Unused
+4	Offset (upper word)	
+6	Offset (lower word)	

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	Returns the current position of the file pointer
	H'0D	

Parameter Block

	One byte	One byte
+0	Return value	File number
+2	Offset (upper word)	
+4	Offset (lower word)	

Parameters

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

The following shows an example for inputting one character as a standard input (from a keyboard). Here the label SYS CALL is specified as a system call address.

```
            MOV.W     #H’0101,R0
            MOV.W     #PARM,R1
            JSR       @SYS CALL
STOP        NOP
SYS_CALL    NOP
PARM        .DATA.W   INBUF
INBUF       .RES.B    2
            .END
```

3.8 Calculating Instruction Execution Cycles

The simulator/debugger calculates the number of instruction execution cycles using the expression defined in the H8S and H8/300 series programming manual and the data bus width and the number of access cycles specified in the **Simulator System** dialog box. However, the resulting number of execution cycles may differ from that for the actual user system because the number of instruction execution cycles for some instructions and processing are treated as follows.

- MOVFPE and MOVTPE instructions

  The number of data transfer cycles of an E-clock-synchronous instruction ranges from 9 to 16. The simulator/debugger calculates the total number of instruction execution cycles by assuming the number of data transfer cycles as 11, and adding the number of operand access cycles to it. The number of operand access cycles is determined by the memory data bus width and the number of memory access cycles.
- EEPMOV instruction

  The number of execution cycles for an EEPROM write instruction is the sum of the instruction read cycles and data transfer cycles.
- SLEEP instruction

  The simulator/debugger does not count the number of execution cycles of the SLEEP instruction because the instruction is usually used to stop program execution.
- Standard I/O and File I/O Processing

  The simulator/debugger does not add the number of these cycles because this function is specific to the simulator/debugger. Note that the standard I/O and file I/O processing is performed during the time interval between the completion of a branch to the location specified by a system call address of BSR and JSR instructions and the return to the caller after file I/O processing.

3.9 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

3.9.1 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
- BREAK_ACCESS: Break based on access to a range of memory
- BREAK_DATA: Break based on the value of data written to memory
- BREAK_REGISTER: Break based on the value of data written to a register
- BREAK_SEQUENCE: Break based on a specified execution sequence

If **[Stop]** is specified as the action for a break condition, user program execution stops when the break condition is satisfied. For details, refer to section 5.1, Break Window.

When a break condition is satisfied and user program execution stops, the instruction at the breakpoint may or may not be executed before a break depending on the type of break, as listed in table 3.3.

Table 3.3 Processing When a Break Condition is Satisfied

Command	Instruction When a Break Condition is Satisfied
BREAKPOINT	Not executed
BREAK_ACCESS	Executed
BREAK_DATA	Executed
BREAK_REGISTER	Executed
BREAK_SEQUENCE	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 on the status bar and execution stops.

3.9.2 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 **Simulator System** dialog box specifies whether to stop or continue the simulation when such an error occurs. Table 3.4 lists the error messages, error causes, and the action of the simulator/debugger in the continuation mode.

Table 3.4 Simulation Errors

Error Message	Error Cause	Processing in Continuation Mode
Address Error	Odd PC value	Operates in the same way as the actual device operation.
	Instruction was fetched from the internal I/O area	
	Access in words from odd-number addresses	
	Access in longwords from odd-number addresses	
Memory Access Error	Access to a memory area that has not been allocated	On memory write, nothing is written; on memory read, all bits are read as 1.
	Write to a memory area having the write protect attribute	
	Read from a memory area having the read disable attribute	
	Access to an area where memory does not exist	
	Write to the EEPROM by instructions other than EEPMOV	
Illegal Instruction	Execution of a code that is not an instruction	Always stops
	Execution of MOV.B Rn, @-SP or MOV.B @SP + Rn	Continues simulation but the result is not guaranteed
Illegal Operation	Incorrect relation between the values in the C flag or H flag of the CCR in the DAA or DAS instruction and the values before adjustment	Continues simulation but the result is not guaranteed
	Zero division executed by the DIVXU or DIVXS instruction, or overflow	

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 3.5 lists the states of the program counter (PC) at simulation error stop. The status register (SR) value does not change at simulation error stop.

Table 3.5 Register States at Simulation Error Stop

Error Message	PC Value
Address Error, Memory Access Error	• When an instruction is read: The start address of the instruction that caused the error. • When an instruction is executed: The instruction address following the instruction that caused the error.
Illegal Instruction	The start address of the instruction that caused the error.
Illegal Operation	The instruction address following 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 continuation mode.

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

3.9.3 Break Due to a Trace Buffer Overflow

After the **[Break]** mode is specified with **[Trace Buffer Full Handling]** in the **Trace Acquisition** dialog box, the simulator/debugger stops execution when the trace buffer becomes full. The following message is displayed in the status bar when execution is stopped.

```
Trace Buffer Full
```

3.9.4 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 in the status bar when execution is stopped.

```
Sleep
```

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

3.9.5 Break Due to the [STOP] Button

Users can forcibly terminate execution by clicking the **[STOP]** button during instruction execution. The following message is displayed in the status bar when execution is terminated.

```
Stop
```

Execution can be resumed with the GO or STEP command.

3.10 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 in the **Set Break** dialog box when the break type is set to **[Break Data]** or **[Break Register]**
- Data in the **Memory** window
- Data in the **Fill Memory** dialog box
- Data in the **Search Memory** dialog box

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

In the simulator/debugger, the rounding mode for floating-point decimal-to-binary conversion can be selected in the **Simulator System** dialog box. One of the following two modes can be selected:

- Round to nearest (RN)
- Round to zero (RZ)

If a denormalized number is specified for binary-to-decimal or decimal-to-binary conversion, it is converted to zero in RZ mode, and it is left as a denormalized number in RN mode. If an overflow occurs during decimal-to-binary conversion, the maximum floating-point value is returned in RZ mode, and the infinity is returned in RN mode.

3.11 Display of Function Call History

The simulator/debugger displays the function call history in the **Stack Trace** window when simulation stops, which enable program execution flow to be checked easily. Selecting a function name in the **Stack Trace** window displays the corresponding source program in the **Source** window; the function that has called the current function can also be checked.

The displayed function call history is updated in the following cases:

- When simulation stops under the break conditions described in section 3.9, Break Conditions.
- When register values are modified while simulation stops due to the above break conditions.
- While single-step execution is performed.

For details, refer to section 5.29, Stack Trace Window.

3.12 Profiler

The simulator/debugger displays the memory address and size allocated to functions and global variables, the number of function calls and the profile data. The displayed contents are as follows.

- Times (the number of times a function was called or a global variable was accessed)
- Cycle (the number of execution cycles)
- Ext_mem (the number of times the external memory was accessed)
- I/O_area (the number of times the internal input/output area was accessed)
- Int_mem (the number of times the internal memory was accessed)

Profile information is displayed in list, tree and chart formats. Using profile information it is possible to optimize user programs by reducing the size and putting the most frequently called functions in-line. Further, using the profile information saved to a file, it is possible to optimize user programs based on operational information using the optimizing linkage editor.

For details, refer to section 5.30, Profile Window (List Sheet), through section 5.32, Profile Chart, and section 4.2.3, Optimize Option PROfile in the Optimizing Linkage Editor Manual.

3.13 Pseudo-Interrupts

The simulator/debugger can generate pseudo-interrupts during simulation in two ways:

1. Pseudo-interrupts generated by break conditions

   A pseudo-interrupt can be generated by using a break command to specify **[Interrupt]** as the action when a break condition is satisfied. For details, refer to section 5.1, Break Window.

2. Pseudo-interrupts generated from the **Trigger** window

   A pseudo-interrupt can be generated by clicking a trigger button in the **Trigger** window. For details, refer to section 5.21, Trigger Window.

If another pseudo-interrupt occurs between a pseudo-interrupt occurrence and its acceptance, only the interrupt that has a higher priority can be processed.

Note: Whether a pseudo-interrupt is accepted is determined by [Priority] setting rather than the vector number. Note, however, that when H'8 or a larger value is specified as the priority, that interrupt is always accepted. The simulator/debugger does not simulate the operation of the interrupt controller.

3.14 Coverage

The simulator/debugger acquires instruction coverage information during instruction execution within the address range specified by the user.

The coverage window displays the following items of the acquired instruction coverage information:

- Times (instruction execution count)
- Pass (result of a conditional branch instruction)

  T: Execution has branched in all cases.

  F: Execution has not branched in any cases.

  T/F: Execution has branched in some cases, but not in others.

  -: The target instruction is not a branch instruction or the instruction has not been executed.
- Address (instruction address)
- Assembler (disassembled display)
- Source (C/C++ or assembly-language source program)

The instruction coverage information can also be displayed in the editor window by highlighting the column corresponding to the source line of the executed instruction.

The instruction coverage information can be saved in or loaded from a file.

The status of each instruction execution can be monitored through the instruction coverage information. In addition, this information can be used to determine which part of a program has not been executed. For details, refer to section 5.39, Coverage Window, through section 5.46, Save Coverage Data Dialog Box.

Section 4 Menus

This document uses the standard Microsoft® menu naming convention.

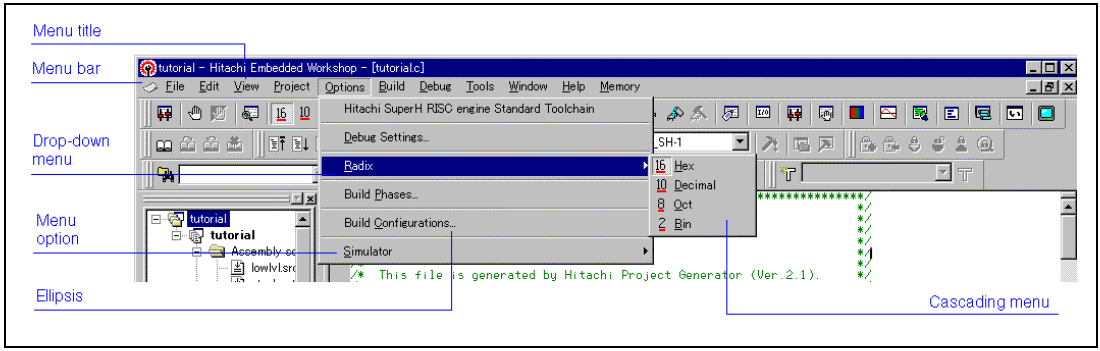


Figure 4.1 Menus

Check marks indicate that the feature provided by the menu option is selected.

Ellipsis indicates that selecting the menu option will open a dialog box that requires extra information to be entered.

Refer to your Windows® user manual for details on how to use the Windows® menu system.

The menu for debugging is described here. For other menus, refer to the HEW user manual.

4.1 View

The View menu is used to select and open new child windows. If the menu option is grayed, then the features provided by the window are not available with the current debugging platform.

4.1.1 Workspace

Opens the **Workspace** window displaying a list of source files.

4.1.2 Output

Opens the **Output** window displaying a message when using the debugger.

4.1.3 Breakpoints

 Opens the **Breakpoints** window allowing the user to view and edit current breakpoints.

4.1.4 Command Line

 Opens the **Command Line** window allowing the user to enter text-based commands to control the debugging platform. These commands can be piped in from a batch file, and the results piped out to a log file, allowing automatic tests to be performed.

4.1.5 Disassembly

 Launches the **Set Address** dialog box allowing the user to enter the address that you wish to view.

4.1.6 IO

 Opens the **IO** window allowing the user to view and modify the control register.

4.1.7 Labels

 Launches the **Labels** window allowing the user to manipulate the current program's symbols (labels).

4.1.8 Locals

 Opens the **Locals** window allowing the user to view and edit the values of the variables defined in the current function. The contents are blank unless the PC is within a C/C++ source-level function.

4.1.9 Memory...

 Launches the **Set Address** dialog box allowing the user to specify a memory block and view format to display within a **Memory** window.

4.1.10 Performance Analysis

 Launches the **Performance Analysis** window allowing the user to set up and view the number of times that particular sections of the user program have been called.

4.1.11 Profile

 Opens the **Profile** window allowing the user to view the address and size of a function or a global variable, the number of times the function is called, and profile data.

4.1.12 Registers

 Opens the **Register** window allowing the user to view all the current CPU registers and their contents.

4.1.13 Status

 Opens the **System Status** window allowing the user to view the debugging platform's current status and the current session and program names.

4.1.14 Trace

 Opens the **Trace** window allowing the user to see the current trace information.

4.1.15 Watch

 Opens the **Watch** window allowing the user to enter C/C++-source level variables and view and modify their contents.

4.1.16 Simulated I/O

 Opens the **Simulated I/O** window enabling the standard I/O and file I/O.

4.1.17 Stack Trace

 Opens the **Stack Trace** window displaying the current stack trace information.

4.1.18 Coverage...

 Opens the **Coverage** window allowing the user to view the coverage information.

4.1.19 Image...

 Opens the **Image** window displaying the memory contents as images.

4.1.20 Waveform...

 Opens the **Waveform** window displaying the memory contents as waveforms.

4.1.21 Trigger

 Opens the **Trigger** window displaying trigger buttons to generate manual interrupts during simulation.

4.2 Options

The Options menu is used to change the settings for the debugging interface of the HEW and make the settings for the debugging platform.

4.2.1 Debug Settings...

Launches the **Debug Settings** dialog box allowing the user to modify the settings for the debugging interface of the HEW (not debugging platform dependent settings).

4.2.2 Radix

Cascades a menu displaying a list of radix in which the numeric values will be displayed and entered by default (without entering the radix prefix). The current radix has a toolbar button to its left is locked down.

For example, if the current radix is decimal then the number ten will be displayed as "`10`" and may be entered as "`10`", "`H'A`", "`0x0a`", etc.; if the current radix is hexadecimal then the number ten will be displayed as "`0A`" and entered as "`A`", "`D'10`", etc.

4.2.3 Simulator

System...:

 Launches a **Simulator System** dialog box allowing the user to modify the debugging platform settings. Refer to section 5.24, Simulator System Dialog Box for more details.

Memory Resource...:

 Opens the **Simulator Memory Resource** window allowing the user to view and edit the debugging platform's current memory map.

4.3 Debug

The Run menu controls the execution of the user program in the debugging platform.

4.3.1 Reset CPU

 Resets the user system hardware and sets the PC to the reset vector address.

4.3.2 Go

 Starts executing the user program at the current PC.

4.3.3 Reset Go

 Executes the user program from the reset vector address.

4.3.4 Go To Cursor

 Starts executing the user program at the current PC and continues until the PC equals the address indicated by the current text cursor (not mouse cursor) position.

4.3.5 Set PC To Cursor

 Changes the value of the Program Counter (PC) to the address at the row of the text cursor (not mouse cursor). Disabled if no address is available for the current row.

4.3.6 Run...

Launches the **Run Program** dialog box allowing the user to enter temporary breakpoints before executing the user program.

4.3.7 Step In

 Executes a block of user program before breaking. The size of this block is normally a single instruction but may be set by the user to more than one instruction or a C/C++-source line (see also section 4.3.10, Step...). If a subroutine call is reached, then the subroutine will be entered and the view is updated to include its code.

4.3.8 Step Over

 Executes a block of user program before breaking. The size of this block is normally a single instruction but can be set by the user to more than one instruction or a C/C++-source line (see also section 4.3.10, Step...). If a subroutine call is reached, then the subroutine will not be entered and sufficient user program will be executed to set the current PC position to the next line in the current view.

4.3.9 Step Out

 Executes sufficient user program to reach the end of the current function and set the PC to the next line in the calling function before breaking.

4.3.10 Step...

Launches the **Step Program** dialog box allowing the user to modify the settings for stepping.

4.3.11 Step Mode

Specifies the **Step Mode** allowing the user to select a unit of stepping from **Auto** (automatic selection), **Assembly** (assembly instruction units), or **Source** (C/C++ source level).

4.3.12 Halt Program

 Stops the execution of the user program.

4.3.13 Initialize

Disconnects the debugging platform and connects it again.

4.3.14 Disconnect

Disconnects the debugging platform.

4.3.15 Download Modules

Downloads the object program.

4.3.16 Unload Modules

Unloads the object program.

4.4 Memory

The Memory menu is used for aspects of the user program that access memory.

4.4.1 Search...

Launches the **Search Memory** dialog box allowing the user to specify the start and end addresses and the data value to be searched and perform the search. Search conditions (match/unmatch and search direction) can also be specified.

4.4.2 Copy...

 Launches the **Copy Memory** dialog box allowing the user to copy a block of the debugging platform's memory to an address within the same memory area. The blocks may overlap, in which case any data within the overlapped region of the source block will be overwritten. If a block of memory is highlighted in a **Memory** window, these will be automatically entered as the start and end addresses when the dialog box is displayed. Data in the source block can be compared with that in the destination while being copied.

4.4.3 Compare...

Launches the **Compare Memory** dialog box, allowing the user to select a start and an end address in the memory area, to check against another area in memory. If a block of memory is highlighted in a **Memory** window, these will be automatically entered as the start and end addresses when the dialog box is displayed.

4.4.4 Fill...

 Launches the **Fill Memory** dialog box allowing the user to fill a block of the debugging platform's memory with a value. If a block of memory is highlighted in a **Memory** window, these will be automatically entered as the start and end addresses when the dialog box is displayed.

4.4.5 Refresh

Forces a manual update of the contents of all open **Memory** windows.

4.4.6 Configure Overlay...

 Launches the **Overlay** dialog box. When the overlay function is used, the target section group can be selected in the dialog box.

Section 5 Windows and Dialog Boxes

This section describes types of windows and dialog boxes, the features that they support and the options available through their associated popup menu.

5.1 Break Window

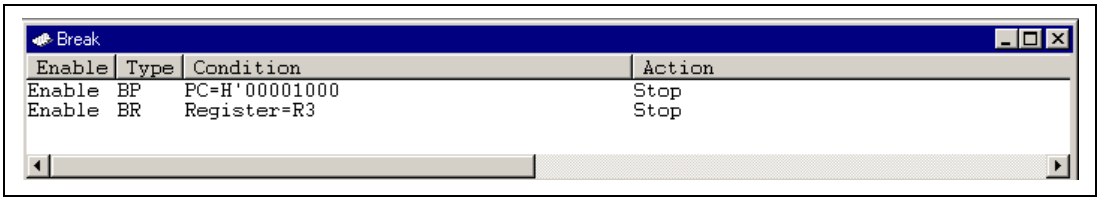


Figure 5.1 Break Window

This window displays all of the specified breakpoints. Items that can be displayed are listed below.

[Enable] Displays whether the breakpoint is enabled or disabled.
Enable: Valid
Disable: Invalid

[Type] Displays break types
BP: PC break
BA: Break access
BD: Break data
BR: Break register (Register name)
BS: Break sequence
BCY: Break cycle

[Condition] Displays the conditions that satisfies a break condition. The contents displayed differ from the type of the break. When the type of the break is BR, the register name is displayed, and when the type of the break is BCY, the number of cycles is displayed.
BP: PC = Program counter (Corresponding file name, line, and symbol name)
BA: Address = Address (Symbol name)
BD: Address = Address (Symbol name)
BR: Register = Register name
BS: PC = Program counter (Corresponding file name, line, and symbol name)
BCY: Cycle = Number of cycles (displayed in hexadecimal)

[Action] Displays the operation of the simulator/debugger when a break condition is satisfied.
Stop: Execution halts
File Input (file name) [File state: Memory data is read from file]
File Output (file name) [File state: Memory data is written to file]

When a breakpoint is double clicked in this window, the **Set Break** dialog box is opened and break conditions can be modified.

A popup menu containing the following options is available by right clicking within the window.

5.1.1 Add...

Sets breakpoints. Clicking this item will open the **Set Break** dialog box and break conditions can be specified.

5.1.2 Edit...

Only enabled when one breakpoint is selected. Select a breakpoint to be edited and click this item. The **Set Break** dialog box will open and break conditions can be changed.

5.1.3 Enable

Enables the selected breakpoint(s).

5.1.4 Disable

Disables the selected breakpoint(s). When a breakpoint is disabled, the breakpoint will remain in the list, but a break will not occur when specified conditions have been satisfied.

5.1.5 Delete

Removes the selected breakpoint. To retain the details of the breakpoint but not have it cause a break when its conditions are met, use the Disable option (see section 5.1.4, Disable).

5.1.6 Delete All

Removes all breakpoints.

5.1.7 Go to Source

Only enabled when one breakpoint is selected. Opens **Source** or **Disassembly** window at address of breakpoint.

5.1.8 Close File

Closes the selected File Input or File Output data file and resets the address to read the file.

5.1.9 Close All Files

Closes all the selected File Input and File Output files and resets the address to read the file.

5.2 Set Break Dialog Box (Condition Sheet)

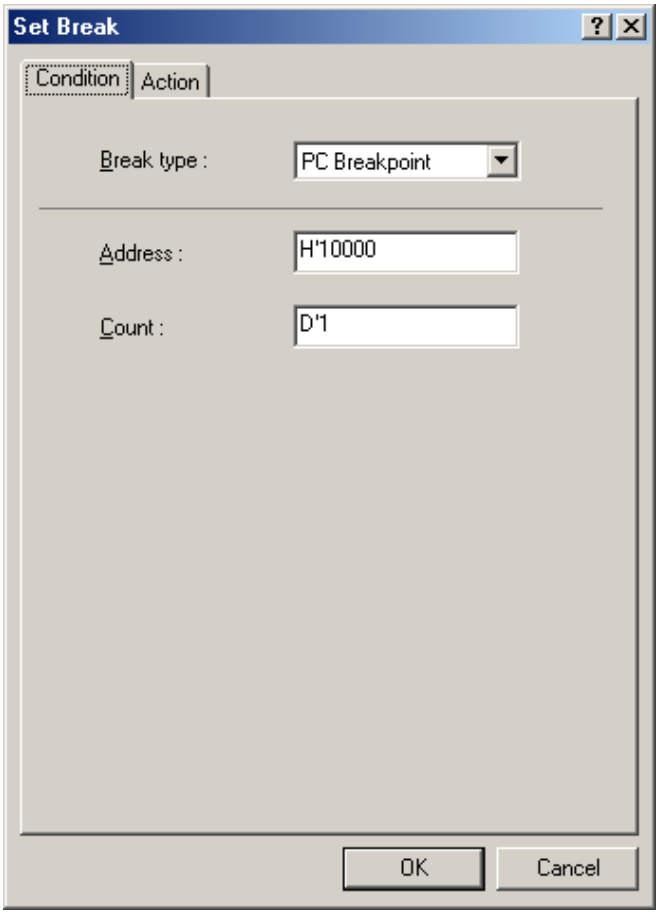


Figure 5.2 Set Break Dialog Box (Condition Sheet)

This dialog box specifies break conditions.

A break type to be set is specified using the radio buttons in the **[Break type]** box. Items that can be specified are listed below.

[PC Breakpoint] Up to 255 breakpoints can be specified
[Address] Address where a break occurs

[Count] Number of times that a specified instruction is fetched (when the prefix is omitted, values must be input and are displayed in decimal) (1 to 16383, default: 1)

[Break Access] Up to two addresses can be specified
- [Start address] Start address of memory where a break occurs if the memory is accessed
- [End address] End address of memory where a break occurs if the memory is accessed (If no data is input, only the start address is break range)
- [Access type] Read, Write, or Read/Write

[Break Data] Up to eight values of data can be specified
- [Start address] Address of memory where a break occurs
- [Data] Data value that causes a break
- [Size] Data size
- [Option] Data match/mismatch

[Break Register] Up to eight register names can be specified
- [Register] Register name where break conditions are specified
- [Size] Data size
- [Data] Data value that causes a break (If no data is input, a break occurs whenever data is written to the register)
- [Option] Data match/mismatch

[Break Sequence] Only one address can be specified
- [Address1] to [Address8] Pass addresses that are the conditions to generate a break. (All eight breakpoints do not have to be set.)

[Break Cycle] Up to 255 cycles can be specified
- [Cycle] Number of cycles to determine a break (H'1 to H'FFFFFFFF). A condition will be satisfied by a number of cycles of **[Cycle]** x n. However, the specified number of cycles and the number of cycles that satisfied the condition may be different.
- [Count] Number of times breaks will occur
  - [ALL] A break will occur whenever a condition is satisfied.
  - [Times] (when the prefix is omitted, values must be input and are displayed in decimal) (1 to 65535)
    A break will occur only when the number of times the conditions is satisfied is equal to or below the value specified for **[Times]**.

When **[PC Breakpoint]** and **[Break Sequence]** is selected, if an overloaded function or class name including a member function is specified in address, the **Select Function** dialog box opens. In the dialog box, select a function. For details, refer to the HEW Debugger User's Manual.

Clicking the **[OK]** button sets the break conditions. Clicking the **[Cancel]** button closes this dialog box without setting the break conditions.

5.3 Set Break Dialog Box (Action Sheet)

Figure 5.3 Set Break Dialog Box (Action Sheet)

This dialog box specifies the operation when a break condition is satisfied. First set the operation in **[Action type]**. The following show the items that can be set in each operation.

[Stop] Stops the operation of user program when a condition is satisfied. There is no item to be set.

[File Input] The data of a specified file is referred to and its contents are written to the specified memory when a condition has been satisfied

- [Input file] Data file to refer to. When the simulator/debugger has reached referring to the end of a file, it will repeat referring to the beginning of the same file.
- [Address] Memory address where data should be written to.
- [Data size] Data size to be written to (1/2/4/8).
- [Count] Number of data to be written to (when the prefix is omitted, values must be input and are displayed in decimal) (H’1 to H’FFFFFFFF).

[File Output] The contents of a specified memory is written to the specified file when a condition has been satisfied.

- [Output file] Data file to be written to.
- [Append] When a existing file is specified for the **[Output File]**,
- [Address] Memory address to read data to.
- [Data size] Size of one data to refer to (1/2/4/8).
- [Count] Number of data to refer to(when the prefix is omitted, values must be input and are displayed in decimal) (H’1 to H’FFFFFFFF).
- [Option] Data match/mismatch

[Interrupt] Interrupts the program when a condition has been satisfied. For details, refer to section 3.13, Pseudo-Interrupts.

- [Interrupt type1] Specifies the interrupt vector number (when the prefix is omitted, values must be input and are displayed in hexadecimal) (H’0 to H’FF)
- [Priority] Interrupt priority (when the prefix is omitted, values must be input and are displayed in hexadecimal) (H’0 to H’11) Whether an interrupt is accepted depends on the specifications of the CPU for the selected debugging platform. When H'8 or a larger value is specified, the interrupt is always accepted.

Note: When the same file is specified for multiple File Inputs, the simulator/debugger will read data from the file in the order conditions are satisfied. When the same file is specified for multiple File Inputs, the simulator/debugger will write data to the file in the order conditions are satisfied. However, when File Input and File Output specify the same file, only the operation for the first condition satisfied is valid.

5.4 Command Line Window

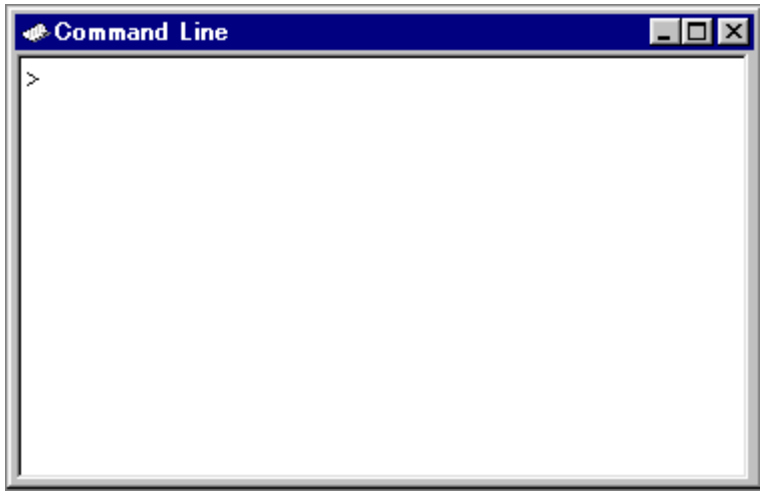


Figure 5.4 Command Line Window

Allows the user to control the debugging platform by sending text-based commands instead of the window menus and commands. It is useful if a series of predefined commands need to be sent to the debugging platform by calling them from a batch file and, optionally, recording the output in a log file. The command can be executed by pressing 'Enter' after the command is input to the last line (or, the **Enter** button in the right of the text box is clicked). For information about the available commands, refer to the on-line help.

If available, the window title displays the current batch and log file names separated by colons.

Pressing the Ctrl + ↑ or Ctrl + ↓ keys on the last line displays the previously executed command line.

Clicking the right mouse button on the **Command Line** window displays the popup menus. The menus include the following options.

5.4.1 Set Batch File...

Launches the **Set Batch File** dialog box, allowing the user to enter the name of a command file (*.hdc). Clicking the **[Play]** button closes the dialog box and the specified command file runs. Clicking the **[OK]** button displays the specified command file name as the window title. Clicking the **[Cancel]** button closes the dialog box without modifying the setting.

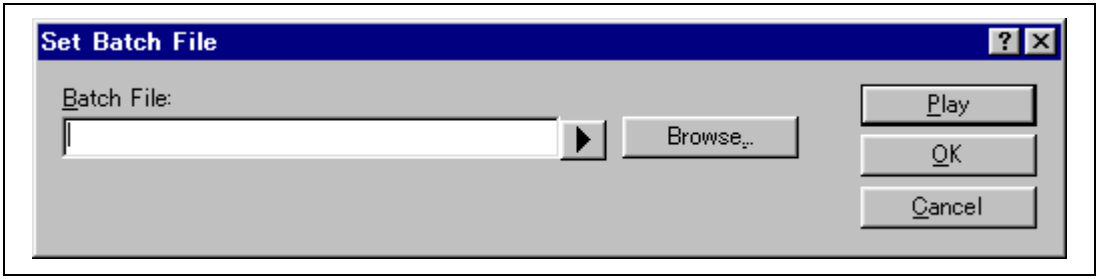


Figure 5.5 Set Batch File Dialog Box

5.4.2 Play

Runs the command file selected in the **Set Batch File** dialog box. It is displayed in a recessed state while the batch file is running and can be used to stop an executing batch file and return control to the user.

5.4.3 Stop

Stops the execution of a command. The button becomes valid during the execution of the command.

5.4.4 Set Log File...

Launches the **Open Log File** dialog box, allowing the user to enter the name of a log file (*.log). The logging option is automatically set and the name of the file shown on the window title bar.

Opening a previous log file will ask the user if they wish to append or over-write the current log.

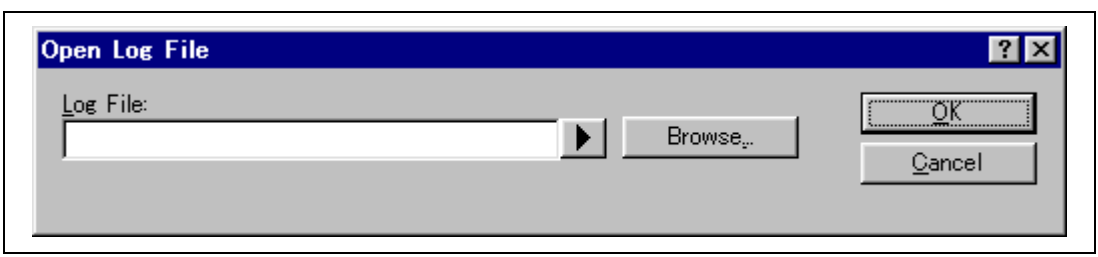


Figure 5.6 Open Log File Dialog Box

5.4.5 Logging

Toggles logging to file on and off. When logging is active, the button becomes effective. Note that the contents of the log file cannot be viewed until logging is completed, or temporarily disabled by clearing the check box. Re-enabling logging will append to the log file.

5.4.6 Browse...

Displays the **Browse** dialog box. This dialog box pastes the full path of the selected file to the cursor location. This option can only be used when the cursor is at the last line.

5.4.7 Placeholder

Pastes the selected placeholder to the cursor location. This function is only available when the cursor is located on the last line.

5.4.8 Select All

Selects all contents output in the **Command Line** window.

5.4.9 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

5.4.10 Paste

This option pastes the contents of the Windows® clipboard to the current cursor location. This option can only be used when the cursor is at the last line.

5.5 Disassembly Window

This window is used to display code at the assembly-language level.

Assembler information is obtained by disassembling the memory contents, and may be edited or viewed directly from memory without requiring debug information from the object file.

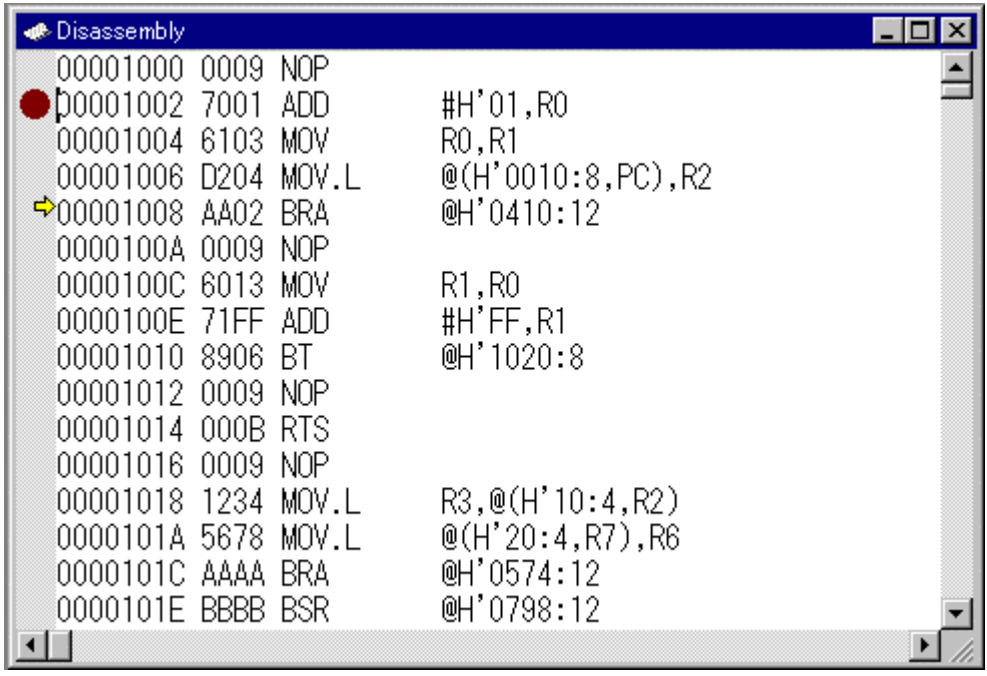


Figure 5.7 Disassembly Window

This window displays address information, addresses, instruction code, and instruction mnemonic. The following are the address information items:

: A bookmark is set.

: A PC Break is set.

: PC location

A popup menu containing the following options is available by right clicking within the window:

5.5.1 View Source

Opens the **Source** window including the source program corresponding to the text cursor (not the mouse cursor) position. Only available when the selected source line is valid.

5.5.2 Go to cursor

Commences to execute the user program starting from the current PC address. The program will continue to run until the PC reaches the address indicated by the text cursor (not the mouse cursor) or another break condition is satisfied. PC breakpoint is used for this function. The function is not available when 255 PC breakpoints have already been specified.

5.5.3 Set Address...

Displays the **Set Address** dialog box. Specify the address from which the display should start.

5.5.4 Set PC Here

Changes the value of the PC to the address indicated by the text cursor (not the mouse cursor).

5.5.5 Edit...

Launches the **Assembler** dialog box allowing the user to modify the instruction at that address. Note that changes to the machine code do not modify the source file, and no changes will be saved when the simulator is terminated.

5.5.6 Code Bytes

Selects whether or not the instruction code is displayed.

5.5.7 Toggle Breakpoint

Enables or disables PC breakpoints.

5.6 IO Window

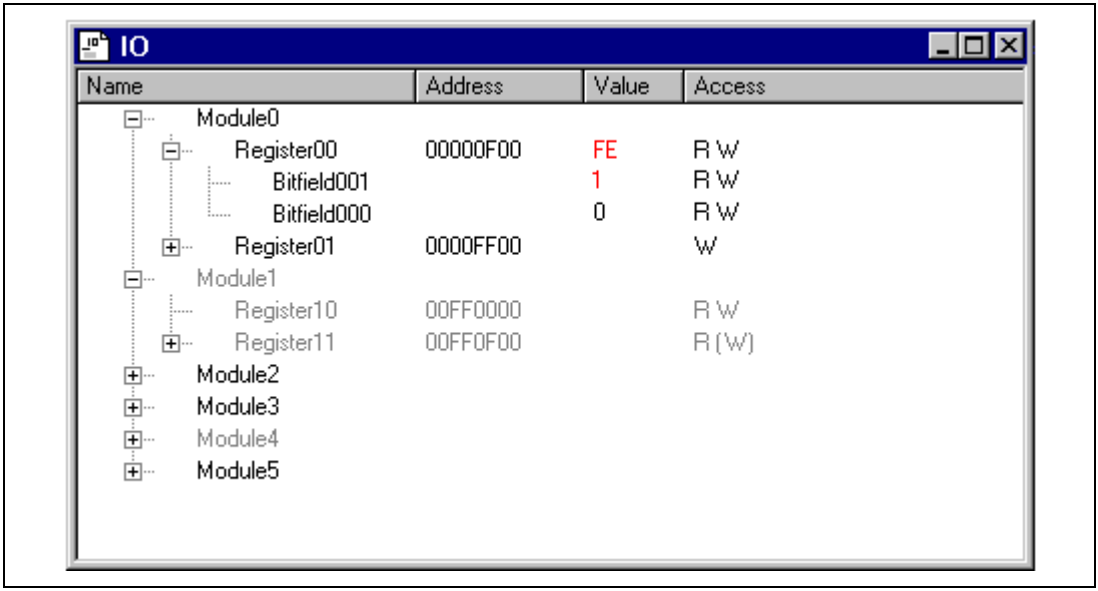


Figure 5.8 IO Window

Refers to and sets the values of control registers.

Double-clicking the plus mark (+) or minus mark (-), or entering the plus key (+) or minus key (-) will expand or compress the information of control registers, and then display it.

Double-clicking the **Name** column displays the Edit Register dialog box. The value of the control registers can be modified.

5.7 Label Window

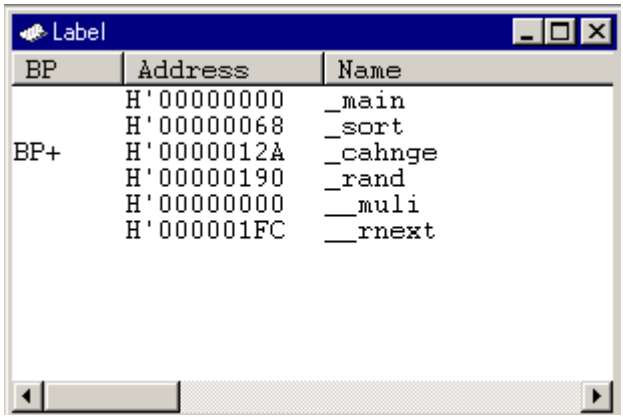


Figure 5.9 Label Window

You can view symbols sorted either alphabetically (by ASCII code) or by address value by clicking on the respective column heading.

It supports column-specific double-click actions:

- BP - Sets or cancels a PC breakpoint at that address.

A popup menu containing the following options is available by right-clicking within the window:

5.7.1 Add...

Launches the **Add Label** dialog box:

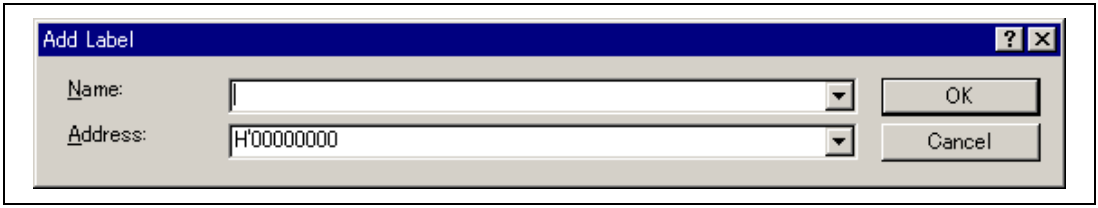


Figure 5.10 Add Label Dialog Box

Enter the new label name into the **Name** field and the corresponding value into the **Address** field and press **[OK]**. The **Add Label** dialog box closes and the label list is updated to show the new label. When an overloaded function or a class name is entered in the **Address** field, the **Select Function** dialog box opens for you to select a function. For details, refer to the HEW Debugger User's Manual.

5.7.2 Edit...

Launches the **Edit Label** dialog box:

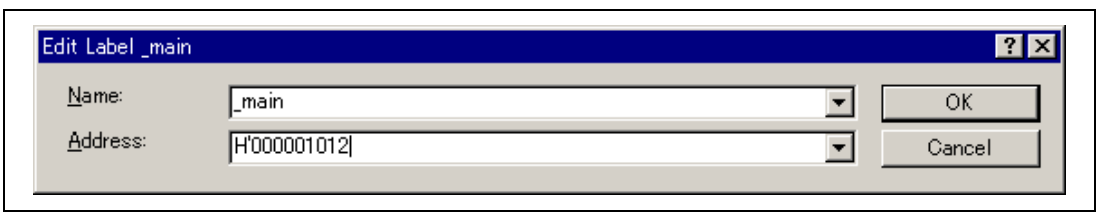


Figure 5.11 Edit Label Dialog Box

Edit the label name and value as required and then press **[OK]** to save the modified version in the label list. The list display is updated to show the new label details. When an overloaded function or a class name is entered in the **Address** field, the **Select Function** dialog box opens for you to select a function. For details, refer to the HEW Debugger User's Manual.

5.7.3 Delete

Deletes the currently selected label from the symbol list. Alternatively use the **Delete** accelerator key. A confirmation message box appears:

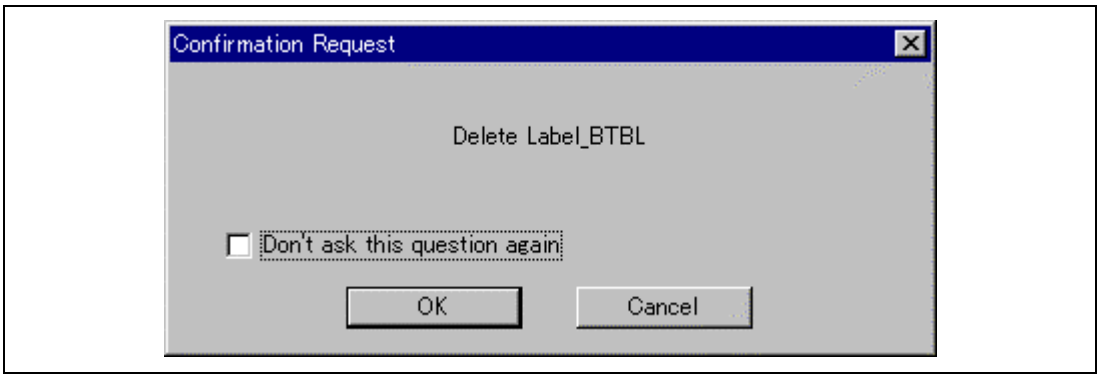


Figure 5.12 Message Box for Confirming Label Deletion

If you click on the **[OK]** button the label is removed from label list and the window display is updated. If the message box is not required then do not select the **Delete Label** option of the **Confirmation** sheet in the **HEW Options** dialog box.

5.7.4 Delete All

Deletes all the labels from the list. A confirmation message box appears:

Figure 5.13 Message Box for Confirming All Label Deletion

If you click on the **[OK]** button all the labels are removed from the HEW system's symbol table and the list display will be cleared. If the message box is not required then do not select the **Delete All Labels** option of the **Confirmation** sheet in the **HEW Options** dialog box.

5.7.5 Load...

Merges a symbol file into HEW's current symbol table. The **Load Symbols** dialog box opens:

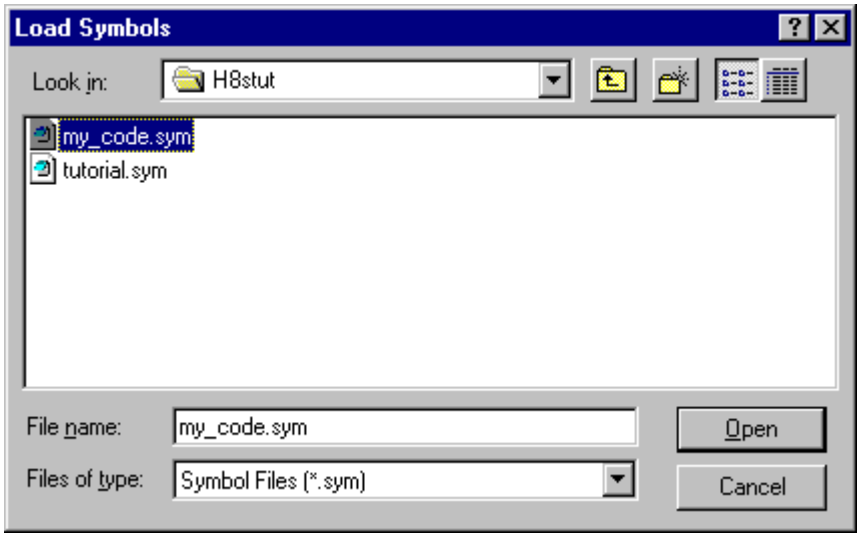


Figure 5.14 Load Symbols Dialog Box

The dialog box operates like a standard Windows® **Open File** dialog box; select the file and click **[Open]** to start loading. The standard file extension for symbol files is ".sym". When the symbol loading is complete a confirmation message box may be displayed showing how many symbols have been loaded (this can be switched off in the **Confirmation** sheet on the **HEW Options** dialog).

5.7.6 Save

Saves HEW's current symbol table to a symbol file.

5.7.7 Save As...

The **Save Symbols** dialog box operates like a standard Windows® **Save File As** dialog box. Enter the name for the file in the **File name** field and click **[Open]** to save HEW's current label list to a symbol file. The standard file extension for symbol files is ".sym".

See appendix B of the HEW Debugger User's Manual for symbol file format.

5.7.8 Find...

Launches the **Find Label Containing** dialog box:

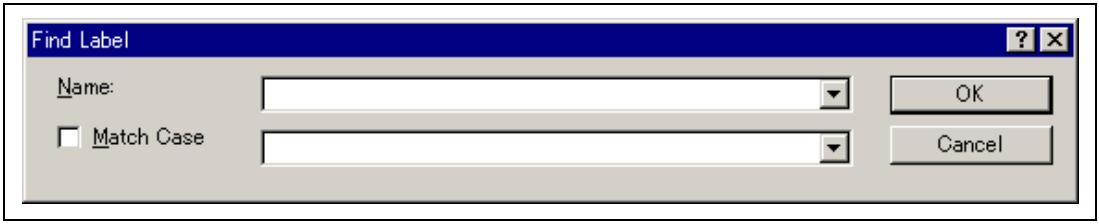


Figure 5.15 Find Label Containing Dialog Box

Enter all or part of the label name that you wish to find into the edit box and click **[OK]** or press **ENTER**. The dialog box closes and HEW searches the label list for a label name containing the text that you entered.

Note: Only the label is stored by 1024 characters of the start, therefore the label name must not overlap mutually in 1024 characters or less. Labels are case sensitive.

5.7.9 Find Next

Finds the next occurrence of the label containing the text that you entered.

5.7.10 View Source

Opens the **Source** or **Disassembly** window containing the address corresponding to the label.

5.8 Locals Window

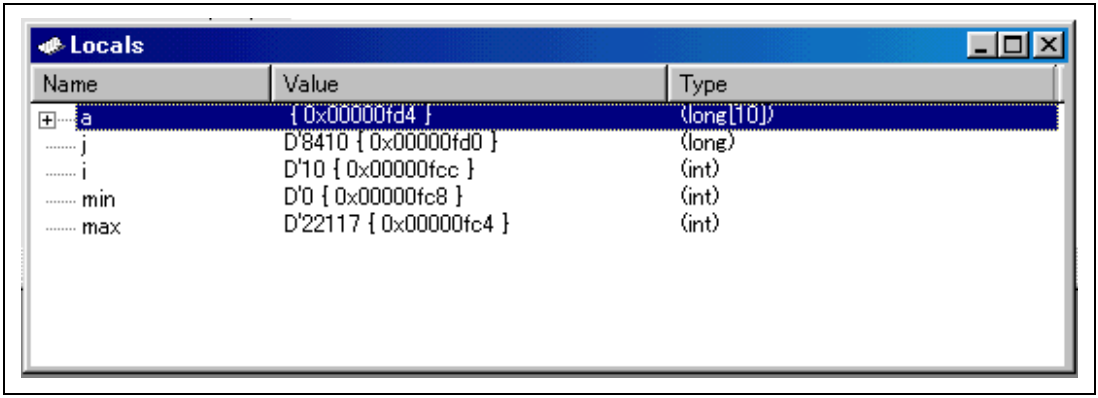


Figure 5.16 Locals Window

Allows the user to view and modify the values of all the local variables. The contents of this window are blank unless the current PC can be associated to a function containing local variables in the source files *via* the debugging information available in the absolute file (*.abs).

The following items are displayed.

[Name] Name of the variable

[Value] Value, assigned location.
The assigned location is enclosed by { }.

[Type] Displays the type of variable

The variables are listed with a plus indicating that the information may be expanded by double-clicking on the variable name, and a minus indicating that the information may be collapsed. Alternatively, the plus and minus keys may be used. For more information on the display of information, refer to the HEW Debugger User's Manual.

A popup menu containing the following options is available by right clicking within the window:

5.8.1 Edit Value...

Launches a dialog box to modify the selected variable's value.

5.8.2 Radix

Changes the radix for the selected local variable display.

5.8.3 Copy

 Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

5.9 Memory Window

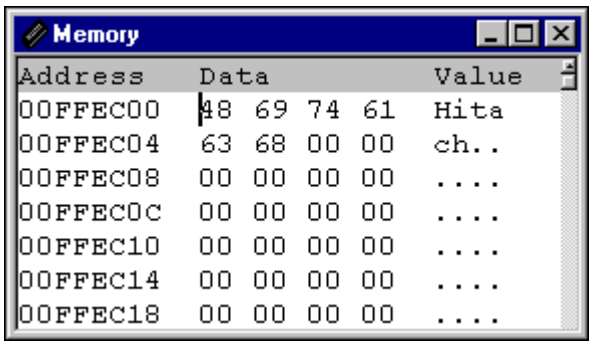


Figure 5.17 Memory Window

Allows the user to view and modify the contents of the debugging platform's memory. Memory may be viewed in ASCII, byte, word, longword, single-precision floating-point, and double-precision floating-point formats, and the title bar indicates the current view style and the address shown as the offset from the previous label (symbol).

The contents of memory can be edited by double-clicking on a data item. The latter will launch the **Edit** dialog box, allowing the user to enter a new value using a complex expression. If the data at that address cannot be modified (i.e. within ROM or guarded memory) then the message "Invalid address value" is displayed.

Double-clicking within the Address column will launch the **Set Address** dialog box, allowing the user to enter an address. Clicking the **[OK]** button will update the window so that the address entered in the **Set Address** dialog box is the first address displayed in the top-left corner.

A popup menu containing the following options is available by right clicking within the window:

5.9.1 Lock Refresh

Controls the **Memory** window so that it is not automatically updated when user program execution stops.

5.9.2 Refresh

Forcibly updates the **Memory** window contents.

5.9.3 Start Address...

Launches the **Set Address** dialog box, allowing the user to enter new start and end addresses. The window will be updated so that this is the first address displayed in the top-left corner. When an overloaded function or a class name including a member function is entered, the **Select Function** dialog box opens for you to select a function. For details, refer to section 7.3, Supporting Duplicate Labels in the HEW Debugger User's Manual. The size to display data can be specified.

5.9.4 Format...

Displays the **Format Memory Display** dialog box. The size to display data, the format to display data, and the font used to display data can be specified.

5.9.5 Search...

Launches the **Search Memory** dialog box, allowing the user to search a block of the debugging platform's memory for a specified data value. If a block of memory is highlighted, the start and end fields in the dialog box will be set automatically with the start and end addresses corresponding to the highlighted block, respectively. Search conditions (match/unmatch and search direction) can also be specified. The **Memory** window is displayed to start with the address which holds the data that satisfies the search conditions.

5.9.6 Search Next

Only available when data has been found with the **Search...** option. This option starts search from the address following the one found with the last search operation.

5.9.7 Copy...

Launches the **Copy Memory** dialog box, allowing the user to copy a block of memory within the debugging platform to another location within the same memory space. The blocks may overlap. The start and end fields may be set similarly to the **Search...** option (see section 5.9.5, Search...). Data in the source block can be compared with that in the destination while being copied.

5.9.8 Compare...

Launches the **Compare Memory** dialog box, allowing the user to select a start and an end address in the memory area, to check against another area in memory. If a block of memory is highlighted

in a **Memory** window, these will be automatically set as the start and end addresses when the dialog box is displayed.

Similar to Verify memory, but compares two blocks in memory.

5.9.9 Fill...

Launches the **Fill Memory** dialog box, allowing the user to fill a block of the debugging platform's memory with a specified value. The start and end fields may be set similarly to the **Search...** option (see section 5.9.5, Search...).

5.9.10 Save...

Launches the **Save Memory As** dialog box, allowing the user to save a block of the debugging platform's memory in an S-Record file (*.mot). The start and end fields may be set similarly to the **Search...** option (see section 5.9.5, Search...).

5.9.11 Load...

Launches the **Load Memory** dialog box, allowing the user to load to the debugging platform's memory from an S-Record file (*.mot) without deleting the current debug information. The offset field may be used to move the address values specified in the file to a different set of addresses. The optional verify flag can be used to check that the information has been downloaded correctly.

5.10 Performance Analysis Window

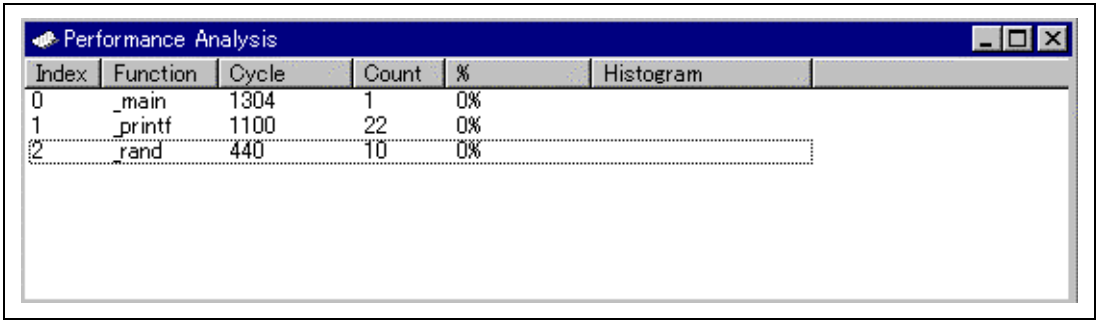


Figure 5.18 Performance Analysis Window

This window displays the number of execution cycles required for the specified functions.

The number of execution cycles can be obtained from the difference between the total number of execution when the target function is called and that when execution returns from the function.

The following items are displayed:

[Index] Index number of the set condition

[Function] Name of the function to be measured (or the start address of the function)

[Cycle] Total number of execution cycles for the function

[Count] Total number of calls for the function

[%] Ratio of execution cycle count required for the function to the execution cycle count required for the whole program

[Histogram] Histogram display of the above ratio

Double-clicking a function to be evaluated displays the **Performance Option** dialog box. In this dialog box, functions can be modified.

A popup menu containing the following options is available by right clicking within the view area:

5.10.1 Add Range...

Adds a new function to be evaluated. Clicking this option launches the **Performance Option** dialog box, allowing the user to add a function. The **Performance Option** dialog box can also be opened by the **Insert** key.

5.10.2 Edit Range

Only enabled when the highlighting bar is on a user-defined range. Launches the **Performance Option** dialog box, allowing the user to modify the range's settings. The **Performance Option** dialog box can also be opened by the **Enter** key.

5.10.3 Reset Counts/Times

Clears the current performance analysis data.

5.10.4 Enable Analysis

Toggles the collection of performance analysis data. When performance analysis is active, a check mark is shown to the left of the text.

5.10.5 Delete Range

Only enabled when the highlighting bar is on a user-defined range. Deletes the range and immediately recalculates the data for the other ranges. The range can also be deleted by the **Delete** key.

5.10.6 Delete All Ranges

Deletes all the current user-defined ranges, and clears the performance analysis data.

5.11 Performance Option Dialog Box

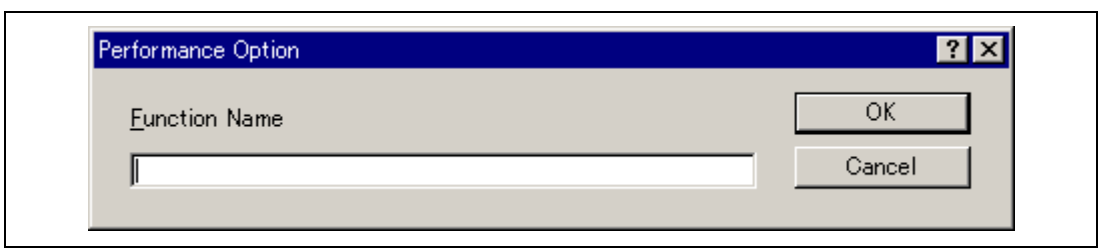


Figure 5.19 Performance Option Dialog Box

This dialog box specifies functions (including labels) to be evaluated. Evaluation results are displayed in the **Performance Analysis** window.

Note that when an overloaded function or a class name including a member function is specified, the **Select Function** dialog box opens. In the dialog box, select a function. For details, refer to section 7.3, Supporting Duplicate Labels of the HEW Debugger User’s Manual.

Clicking the **[OK]** button stores the setting. Clicking the **[Cancel]** button closes this dialog box without setting the function to be evaluated.

5.12 Register Window

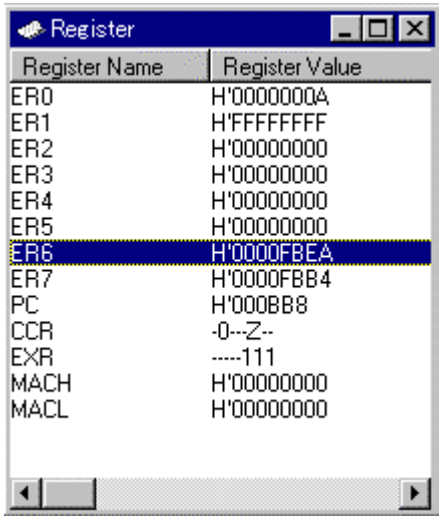


Figure 5.20 Register Window

Allows the user to view and modify the current register values.

A popup menu containing the following options is available by right clicking within the window:

5.12.1 Edit...

Launches the **Edit Register** dialog box, allowing the user to set the value of the register indicated by the text cursor (not mouse cursor).

5.13 Source Window

The **Source** window can be used to view any source file that was included within the object file's debug information - this may be C/C++ and assembly language. This window also displays the coverage information.

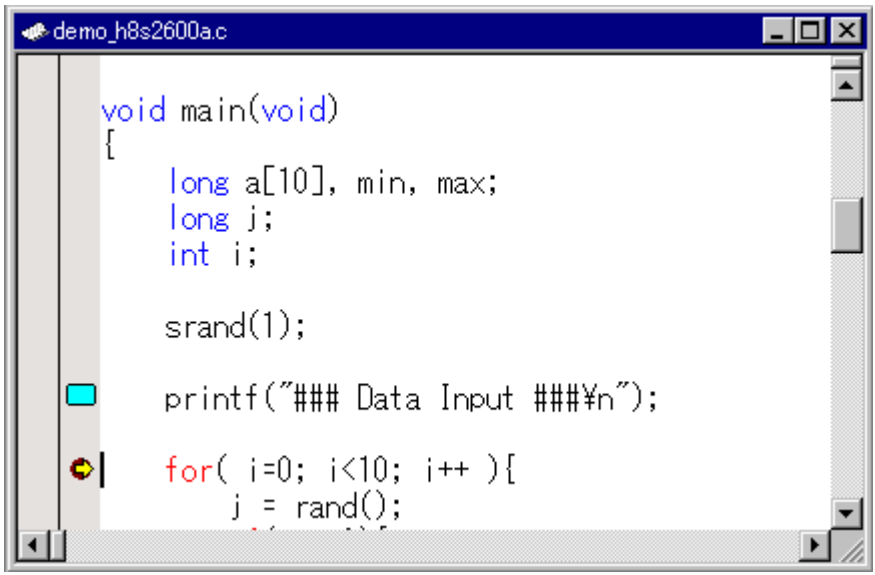


Figure 5.21 Source Window

In this window, the following items are shown on the left as line information.

: A bookmark is set.

: A PC Break is set.

: PC location

A popup menu containing the following options is available by right clicking within the window:

5.13.1 Toggle Breakpoint

Sets or removes PC breakpoints.

5.13.2 Enable/Disable Breakpoint

Enables or disables PC breakpoints.

5.13.3 Instant Watch...

Opens the **Instant Watch** dialog box and displays the variable at the cursor location.

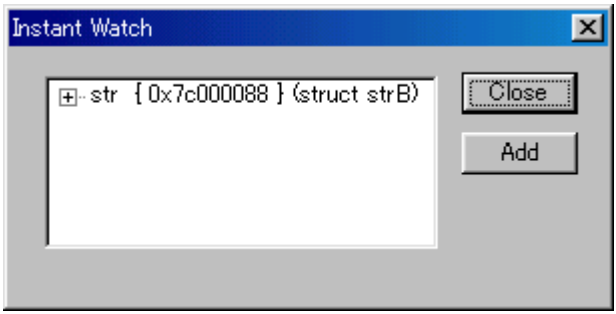


Figure 5.22 Instant Watch Dialog Box

"+" shown to the left of the variable name indicates that the information may be expanded by clicking on the variable name, and "-" indicates that the information may be collapsed. Clicking **[Add]** registers the variable in the **Watch** window and closes the dialog box. Clicking **[Close]** closes the dialog box without registering the variable in the **Watch** window.

5.13.4 Go To Cursor

Commences to execute the user program starting from the current PC address. The program will continue to run until the PC reaches the address indicated by the text cursor (not the mouse cursor) or another break condition is satisfied. PC breakpoint is used for this function. The function is not available when 255 PC breakpoints have already been specified.

5.13.5 Set PC Here

Changes the value of the PC to the address indicated by the text cursor (not the mouse cursor).

5.13.6 Go To Disassembly

Opens a **Disassembly** window showing code disassembled from the address that corresponds to the current line of source code.

5.14 Source Address Column

When a program is downloaded, the **Source** window display is updated to the current source file addresses. The source file addresses are displayed in the left part of the **Source** window. These

addresses are helpful when setting the PC value or a breakpoint. Figure 5.23 shows the **Source** window and the address column.

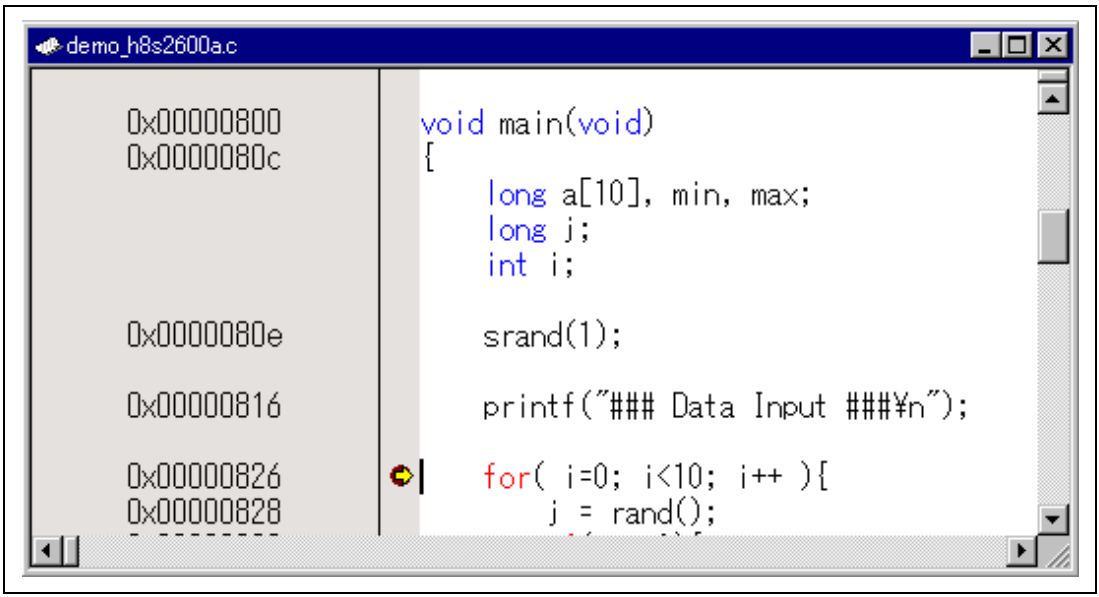


Figure 5.23 Source Window and Address Column

Use the following procedure to display or close the address column for all files:

1. Right-click the mouse in the editor window.
2. Click the **[Define Column Format...]** menu item.
3. The **Global Editor Column States** dialog box will appear.
4. Each check box specifies whether the corresponding column is displayed. If the check box is selected, the column is displayed. If the check box is shaded, the column is displayed for some files and not displayed in the other files.
5. Click the **[OK]** button to update the display with the new settings.

For a single file, use the following procedure to display or close the address column:

1. Right-click the mouse in the editor window of the column that needs to be changed, and an editor popup menu will appear.
2. Click the **[Column]** menu item to display a list of the column names. When a check mark is shown to the left of the column name, the column is displayed. Clicking on a column name can display or close the column.

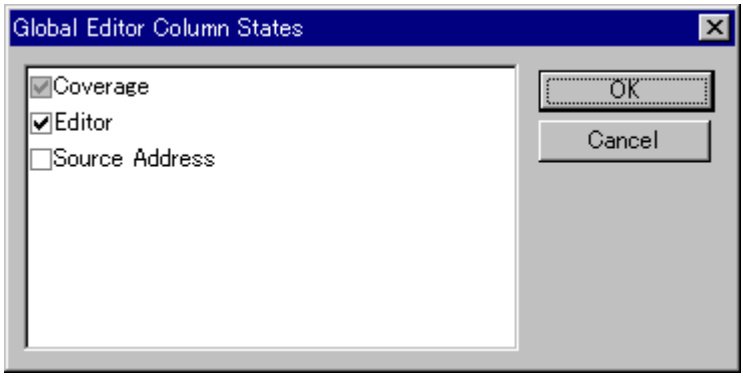


Figure 5.24 Global Editor Column States Dialog Box

5.15 Debugger Column

Each component can add a specific column in the **Disassembly** and **Source** windows. A typical example of a debugger column is the coverage column that displays the code coverage in a graphical form during debugging. Another example is the target component column that shows the hardware breakpoints set in the user system.

Right-clicking on a column displays a popup menu specifically for that column. The menu items differ between columns. The operation when a column is double-clicked depends on the column. For example, double-clicking on the target column will specify a hardware breakpoint.

5.16 Status Window

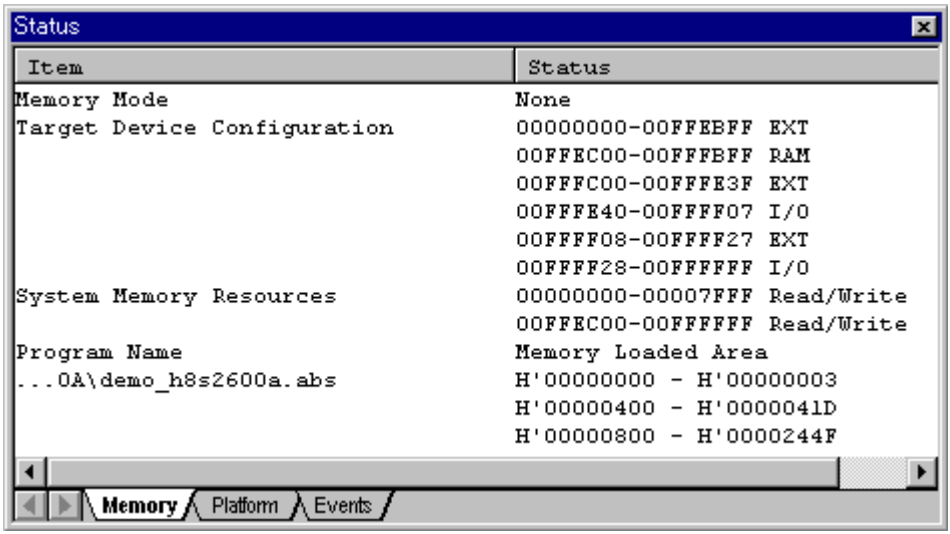


Figure 5.25 Status Window

Allows the user to view the current status of the debugging platform.

The **Status** window has three panes:

1. Memory - contains information about the current memory status including the memory mapping resources and the areas used by the currently loaded object file.
2. Platform - contains information about the current status of the debugging platform, typically including CPU type and mode; and run status.
3. Events - contains information about the current event (breakpoint) status, including resource information.

5.17 Trace Window

This window displays trace information. The displayed information items depend on the target CPU. The trace acquisition conditions can be specified in the **Trace Acquisition** dialog box.

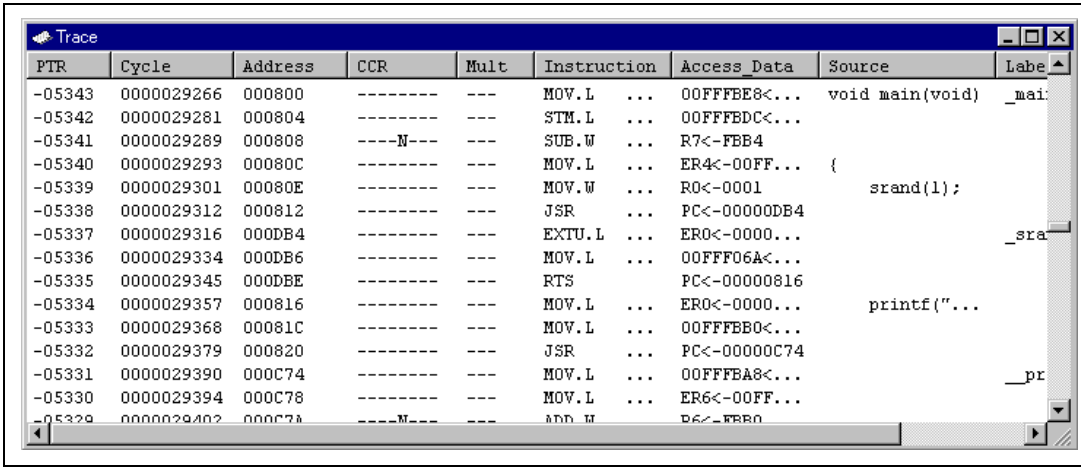


Figure 5.26 Trace Window

This window displays the following trace information items:

[PTR] Pointer in the trace buffer (0 for the last executed instruction)

[Cycle] Total number of instruction execution cycles (cleared by pipeline reset)

[Address] Instruction address

[CCR] The contents of the condition code register (CCR) are displayed in a mnemonic form.

[Mult] The flags in the multiplier are displayed in a mnemonic form (only for the H8S/2600 series).

[Instruction] Instruction mnemonic

[Access Data] Data access (displayed in the form of [Transfer destination <- Transfer data])

[Source] C/C++ or assembly-language source programs

Double-clicking a line in the **Trace** window opens the **Source** window or **Disassembly** window. In the window, the source code is displayed and the selected line is indicated by the cursor.

A popup menu containing the following options is available by right clicking within the window:

5.17.1 Find...

Launches the **Trace Search** dialog box, allowing the user to search the current trace buffer for a specific trace record.

5.17.2 Find Next

If a find operation is successful, and the item found is non-unique, then this will move to the next similar item.

5.17.3 Acquisition...

Launches the **Trace Acquisition** dialog box, allowing the user to define the area of user program to be traced.

5.17.4 Clear

Empties the trace buffer in the debugging platform. If more than one trace window is open, all **Trace** windows will be cleared as they all access the same buffer.

5.17.5 Save...

Launches the **Save As** file dialog box, allowing the user to save the contents of the trace buffer as a text file. A range can be defined based on the PTR number (saving the complete buffer may take several minutes). Note that this file cannot be reloaded into the trace buffer.

5.17.6 View Source

Displays the source program corresponding to the address.

5.17.7 Trim Source

Removes white space from the left side of the source.

5.17.8 Statistic

Analyzes statistical information under the specified conditions.

5.18 Trace Acquisition Dialog Box

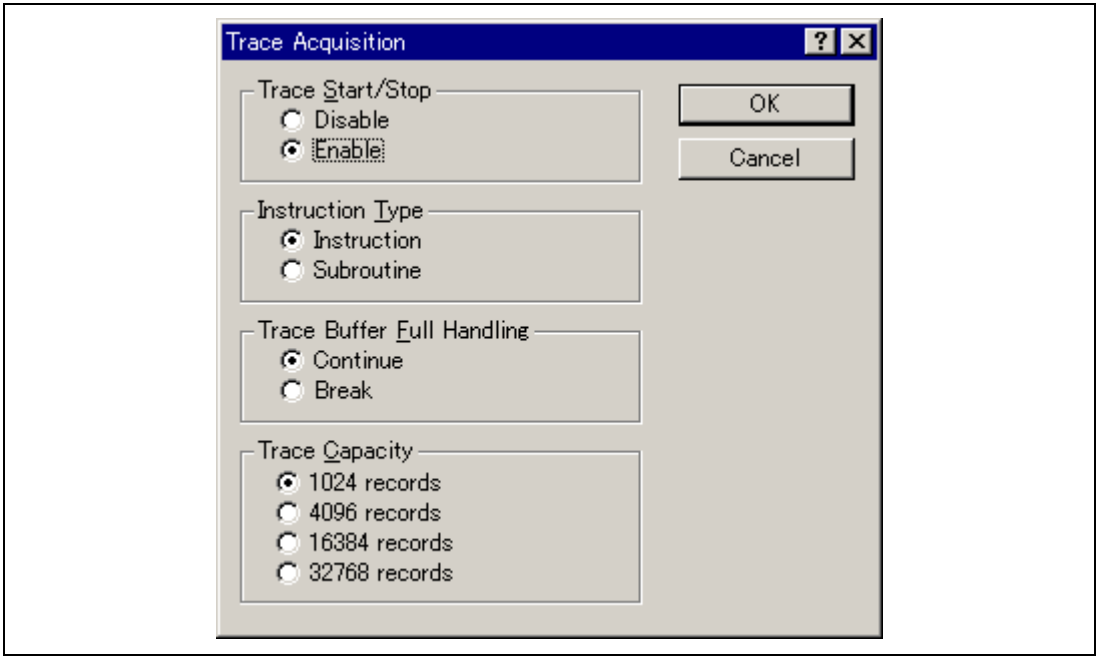


Figure 5.27 Trace Acquisition Dialog Box

This dialog box specifies the conditions for trace information acquisition.

[Trace Start/Stop]

[Disable]	Disables trace information acquisition.
[Enable]	Enables trace information acquisition.

[Instruction Type]

[Instruction]	Acquires trace information for all instructions.
[Subroutine]	Acquires trace information for the subroutine instructions only.

[Trace Buffer Full Handling]

[Continue]	Continues acquiring trace information even if the trace information acquisition buffer becomes full.
[Break]	Stops execution when the trace information acquisition buffer becomes full.

The trace buffer capacity can be selected from 1024 records, 4096 records, 16384 records, and 32768 records in the **[Trace Capacity]**.

Clicking the **[OK]** button stores the settings. Clicking the **[Cancel]** button closes this dialog box without modifying the settings.

5.19 Trace Search Dialog Box

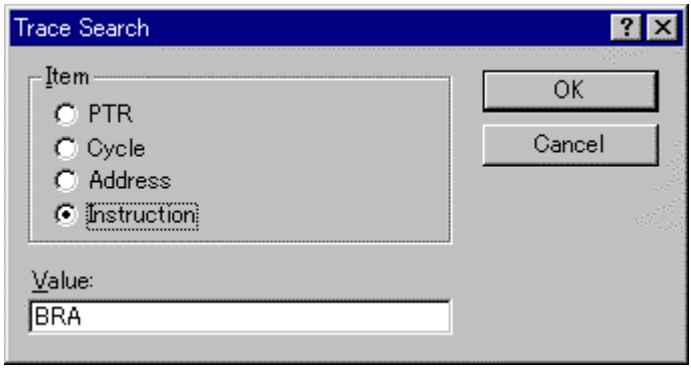


Figure 5.28 Trace Search Dialog Box

This dialog box specifies the conditions for searching trace information. Specify a search item in **[Item]** and search for the specified contents in **[Value]**.

[PTR]	Pointer in the trace buffer (0 for the last executed instruction, specify in the form of -nnn)
[Cycle]	Total number of instruction execution cycles
[Address]	Instruction address
[Instruction]	Instruction mnemonic

Clicking the **[OK]** button stores the settings. Clicking the **[Cancel]** button closes this dialog box without searching.

5.20 Trace Statistic Dialog Box

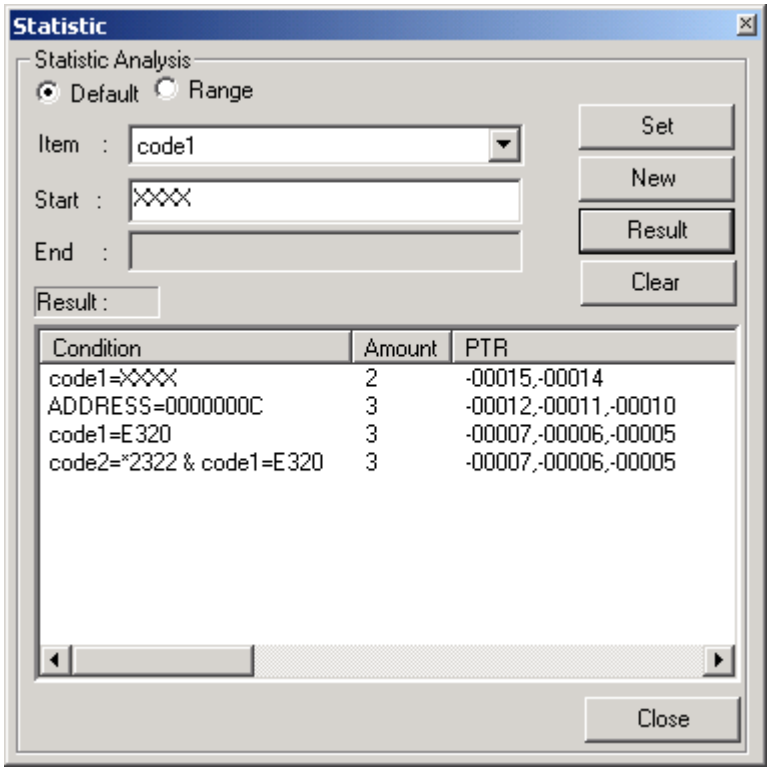


Figure 5.29 Trace Statistic Dialog Box

Allows the user to analyze statistical information concerning the trace data. The target of analysis is specified in **[Item]** and the input value or character string is specified by **[Start]** and **[End]**.

When **[Default]** is selected, the input value or character string cannot be specified as a range. To specify a range, select **[Range]**.

[Set] Adds a new condition to the current one

[New] Creates a new condition

[Result] Obtains the result of statistical information analysis

[Clear] Clears all condition and results of statistical information analysis

Clicking the **[Close]** button closes this dialog box.

5.21 Trigger Window

Displays trigger buttons to manually generate interrupts. The details of the interrupt to be generated by pressing each trigger button can be specified in the **Trigger Setting** dialog box.

Up to 16 trigger buttons can be used.

For details on the interrupt processing in the simulator/debugger, refer to section 3.13, Pseudo-Interrupts.

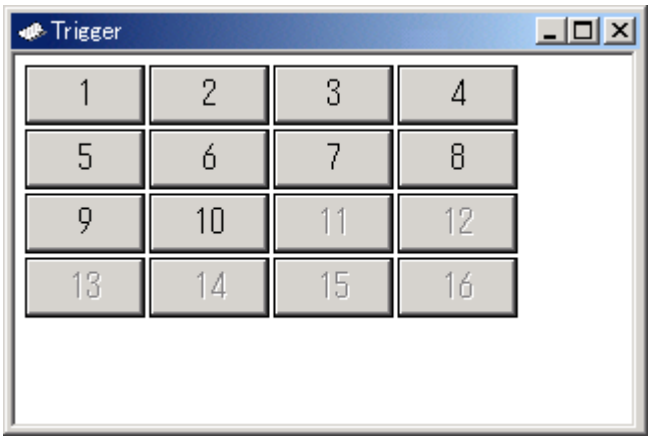


Figure 5.30 Trigger Window

A popup menu containing the following options is available by right-clicking the mouse within the window:

5.21.1 Setting...

Launches the **Trigger Setting** dialog box, allowing the user to specify the details of the interrupt to be generated by pressing each trigger button.

5.21.2 Size

Specifies the size of trigger buttons displayed in the **Trigger** window. **Large**, **Normal**, or **Small** can be selected from the submenu.

5.22 Trigger Setting Dialog Box

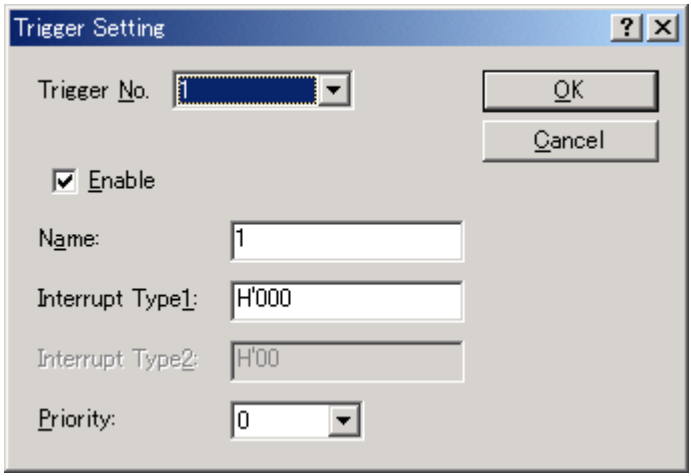


Figure 5.31 Trigger Setting Dialog Box

Allows the user to specify the details of the interrupt to be generated by pressing each trigger button.

[Trigger No.] Selects a trigger button to specify details

[Name] Specifies the name of the selected trigger button, which will be displayed in the **Trigger** window

[Enable] Enables the trigger button when this box is checked.

[Interrupt Type1] Specifies an interrupt vector number (H'0 to H'FF)

[Priority] Interrupt priority (when the prefix is omitted, values must be input and are displayed in hexadecimal) (H’0 to H’11)
Whether an interrupt is accepted depends on the specifications of the CPU for the selected debugging platform. When H'8 is specified, the interrupt is always accepted.

Clicking the **[OK]** button stores the setting. Clicking the **[Cancel]** button closes this dialog box without setting the details of the interrupt.

Note: If the [Cancel] button is clicked after multiple trigger button settings are modified, the modifications of all those buttons are canceled.

5.23 Watch Window

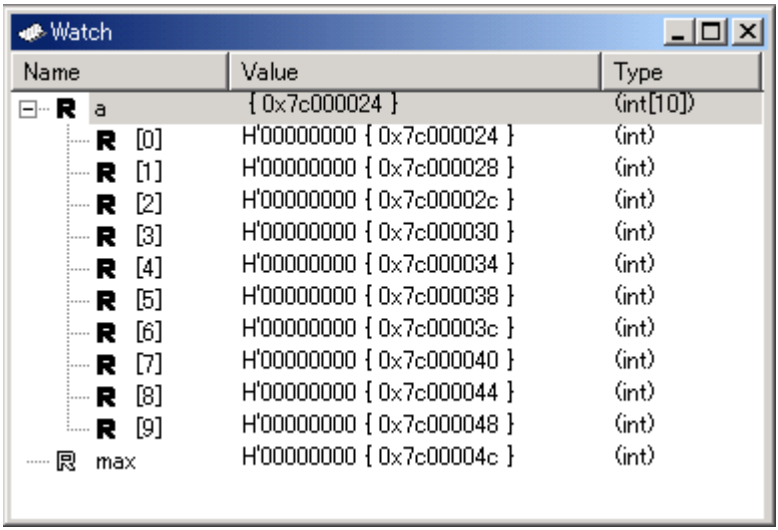


Figure 5.32 Watch Window

Allows the user to view and modify C/C++-source level variables. The contents of this window are displayed only when the debugging information available in the absolute file (*.abs) includes the information on the C/C++ source program. The variable information is not displayed if the source program information is excluded from the debugging information during optimization by the compiler. In addition, the variables that are declared as macro cannot be displayed.

The following items are displayed.

[Name] Name of the variable

[Value] Value, assigned location, and type.
The assigned location is enclosed by { }.

[Type] Displays the type of variable

“+” shown to the left of the variable name indicates that the information may be expanded by double-clicking on the variable name, and “-” indicates that the information may be collapsed. Alternatively, the “+” and “-” keys may be used.

“R” specifies whether the corresponding variable is updated in real time. When a “R” is bold, the value of the corresponding variable will be updated in real time during user program execution.

A popup menu containing the following options is available by right-clicking within the window:

5.23.1 Auto Update

“R” mark of the selected variable becomes bold-faced type and updates the variable in real time.

5.23.2 Auto Update All

All “R” marks become bold-faced type and update all the displayed variables in real time.

5.23.3 Delete Auto Update

“R” mark of the selected variable becomes not bold-typed face and cancels real time update.

5.23.4 Delete Auto Update All

All “R” marks become not bold-faced type and cancel real time update.

5.23.5 Add Watch...

Launches the **Add Watch** dialog box, allowing the user to enter a variable to be watched.

5.23.6 Edit Value...

Launches the **Edit Value** dialog box, allowing the user to change the variable's value. Particular care should be taken when the value of a pointer is changed as it may no longer point to valid data.

5.23.7 Delete

Removes the selected variable from the **Watch** window.

5.23.8 Delete All

Removes all the variables from the **Watch** window.

5.23.9 Radix

Modifies the radix for the selected variable display.

5.23.10 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

5.23.11 Save As...

Launches the **Save As** dialog box, allowing the user to specify the name of a file and saves the contents of the **Watch** window in the file. If the **Append** check box is selected, the window contents are appended to the existing file, and if it is not selected, the existing file is overwritten.

5.23.12 Go To Memory...

Displays the memory contents to which the selected variable is assigned in the **Memory** window.

5.24 Simulator System Dialog Box

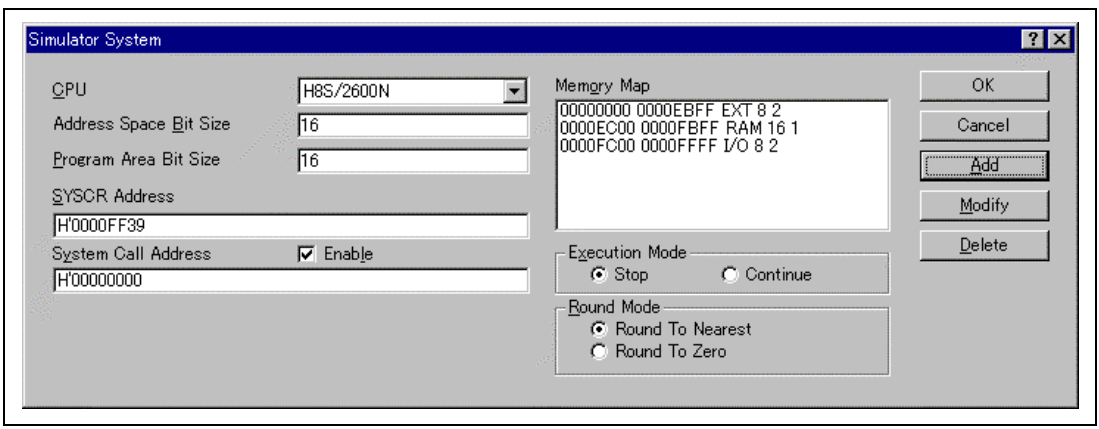


Figure 5.33 Simulator System Dialog Box

This dialog box specifies the width, in bits, of the address space and program area, system call start location, execution mode, floating-point rounding mode, and memory map.

[CPU] Displays the current CPU. (The CPU must be specified in the **Debug Setting** dialog box.)

[Address Space Bit Size] Specifies the width, in bits, of the address space for the selected CPU:
H8/300, H8/300L, H8/300HN, H8S/2600N, H8S/2000N: 16
H8/300HA : 17, 24
H8S/2600A, H8S/2000A : 17, 32

[Program Area Bit Size] Specifies the width, in bits, of the program area for each CPU:
H8/300, H8/300L, H8/300HN, H8S/2600N, H8S/2000N: 16

H8/300HA : Same as the width of the address space

H8S/2600A, H8S/2000A : 17, 24

[System Call Address] Specifies the start address of a system call that performs standard input/output or file input/output processing from the user system.
[Enable] Specifies whether the system call is enabled or disabled.

[Execution Mode] Specifies whether the simulator/debugger stops or continues operating when a simulation error occurs.
[Stop] Stops the simulation.
[Continue] Continues the simulation.

[Round Mode] Specifies the rounding mode for floating-point decimal-to-binary conversion.
[Round to nearest] Rounds to the nearest value.
[Round to zero] Rounds toward zero.

In the **[Memory Map]**, the start address, end address, memory type, data bus width, and access cycles are displayed in that order.

[Memory Map] can be specified, modified, or deleted using the following buttons:

[Add] Specifies **[Memory Map]** items. Clicking this button opens the **Memory Map Modify** dialog box, and memory map items can be specified.

[Modify] Modifies **[Memory Map]** items. Select an item to be modified in the list box and click the **[Modify]** button. The **Memory Map Modify** dialog box opens and memory map items can be modified.

[Delete] Deletes **[Memory Map]** items. Select an item to be deleted in the list box and click the **Delete** button.

Clicking the **[OK]** button stores the modified settings. Clicking the **[Cancel]** button closes this dialog box without modifying the settings.

5.25 Memory Map Modify Dialog Box

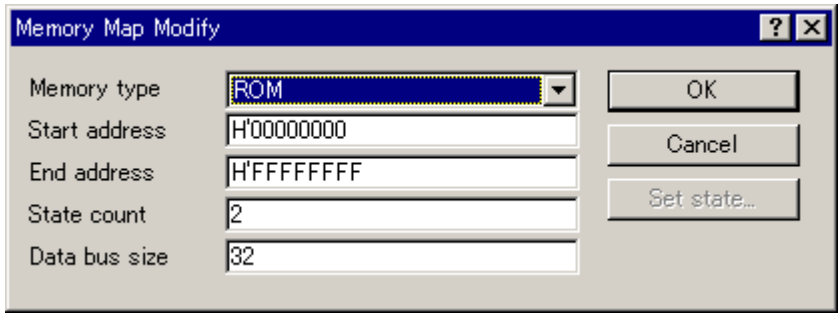


Figure 5.34 Memory Map Modify Dialog Box

This dialog box specifies the memory map of the target CPU of the simulator/debugger.

The contents displayed in this dialog box depend on the target CPU. The specified data is used to calculate the number of cycles for memory access.

[Memory type] Memory type

[Start address] Start address of the memory corresponding to a memory type

[End address] End address of the memory corresponding to a memory type

[State count] Number of memory access cycles

[Data bus size] Memory data bus width

Clicking the **[OK]** button stores the settings. Clicking the **[Cancel]** button closes this dialog box without modifying the settings.

Note: The memory map setting for the area allocated to a system memory resource cannot be deleted or modified. First delete the system memory resource allocation with the Simulator Memory Resource dialog box, then delete or modify the memory map setting.

5.26 Simulator Memory Resource Dialog Box

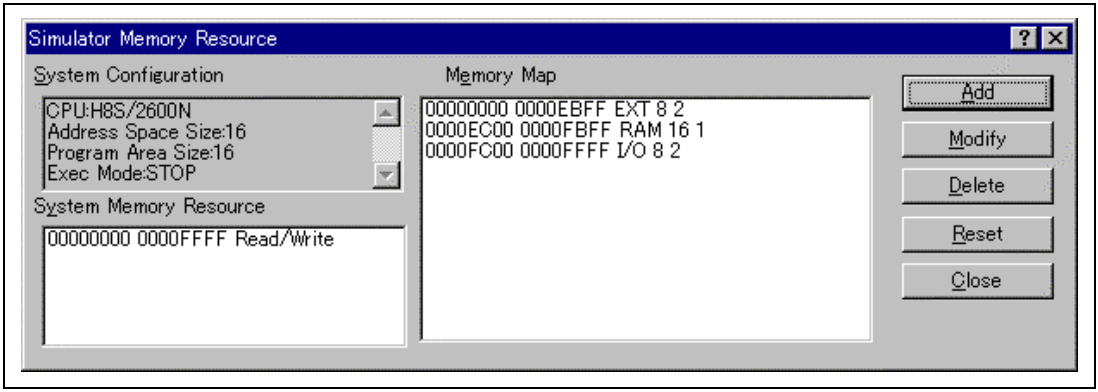


Figure 5.35 Simulator Memory Resource Dialog Box

This dialog box sets and modifies a memory map and information on the target CPU.

[System Configuration] Displays the target CPU, address bus width, and execution mode of the simulator/debugger.

[System memory resource] Displays the access type, start address, and end address of the current memory resources.

[Memory map] Displays the start address, end address, memory type, data bus width, and access cycles.

[System memory resource] can be specified, modified, and deleted using the following buttons:

[Add] Specifies **[System memory resource]** items. Clicking this button opens the **System Memory Resource Modify** dialog box, and **[System memory resource]** items can be specified.

[Modify] Modifies **[System memory resource]** items. Select an item to be modified in the list box and click this button. The **System Memory Resource Modify** dialog box opens and **[System memory resource]** items can be modified.

[Delete] Deletes **[System memory resource]** items. Select an item to be deleted in the list box and click this button.

Note that the **[Reset]** button can reset the **[Memory map]** and **[System memory resource]** to the default value. Clicking the **[Close]** button closes this dialog box. **[System memory resource]** contains the same setting information as **[Memory resource]** on the **Simulator** sheet in the **H8S, H8/300 Standard Toolchain** dialog box. When information about either of the items **[System memory resource]** or **[Memory resource]** is modified, the other item is modified as well.

For details on the Hitachi H8S, H8/300 Standard Toolchain dialog box, refer to section 2.3.1, Memory Mapping in the HEW Debugger User's Manual.

5.27 System Memory Resource Modify Dialog Box

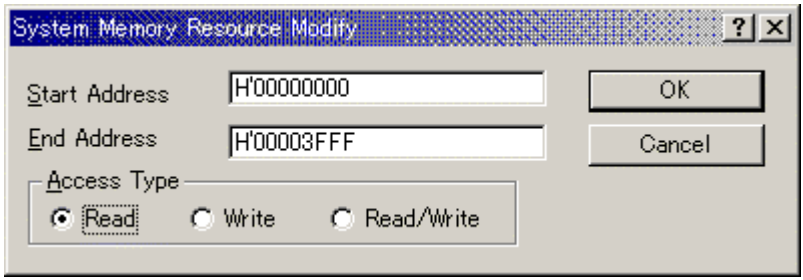


Figure 5.36 System Memory Resource Modify Dialog Box

This dialog box specifies or modifies system memory settings.

[Start address] Start address of the memory area to be allocated

[End address] End address of the memory area to be allocated

[Access type] Access type

Read:	Read only
Write:	Write only
Read/Write:	Read and write

Click the **[OK]** button after specifying the **[Start address]**, **[End address]**, and **[Access type]**. Clicking the **[Cancel]** button closes this dialog box without modifying the setting.

5.28 Simulated I/O Window

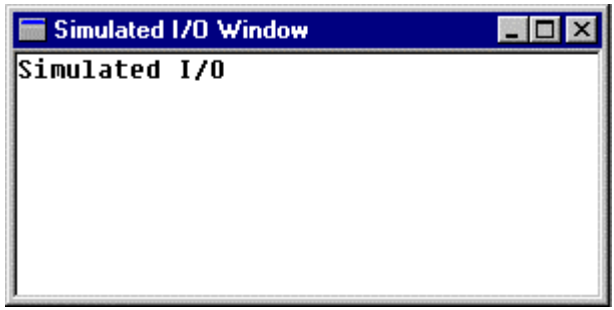


Figure 5.37 Simulated I/O Window

This window is for standard I/O and file I/O system calls from the user program.

Clicking the right mouse button on the **Simulated I/O** window displays the following popup menus.

[Copy] Copies the highlighted text to the Windows® clipboard so that the text can be pasted to another application.

[Paste] Pastes the text from the Windows® clipboard to the **Simulated I/O** window.

[Clear Window] Clears the contents of the **Simulated I/O** window.

For the I/O processing, refer to section 3.7, Standard I/O and File I/O Processing.

5.29 Stack Trace Window

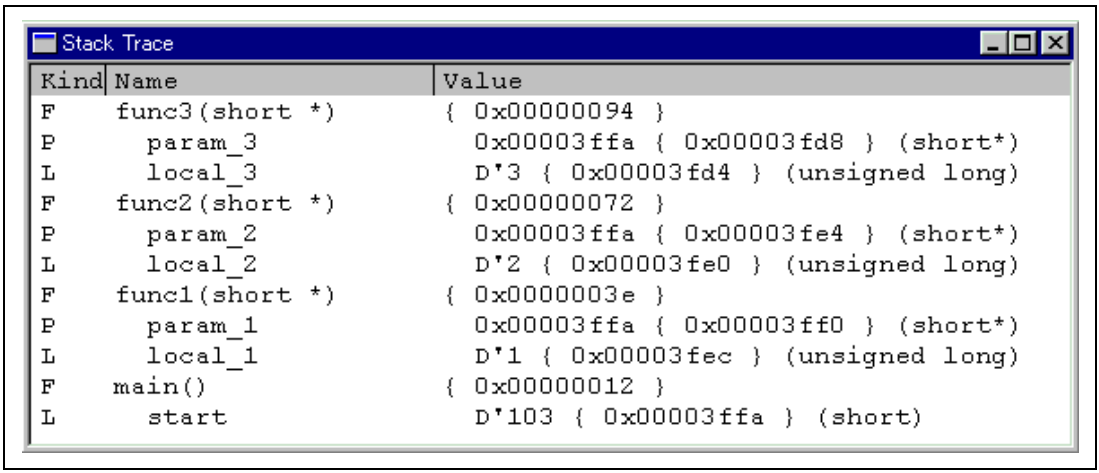


Figure 5.38 Stack Trace Window

This window displays the function call history.

The following items are displayed.

[Kind] Indicates the type of the symbol.
F: Function
P: Function parameter
L: Local variable

[Name] Indicates the symbol name.

[Value] Indicates the value, address, and type of the symbol.

Right-clicking on the mouse within the window displays a popup menu. Supported menu options are described in the following sections:

5.29.1 Go to Source

Displays, in the **Source** window, the source program corresponding to the selected function.

5.29.2 View Setting...

Launches the **Stack Trace Setting** dialog box, allowing the user to specify the **Stack Trace** window settings.

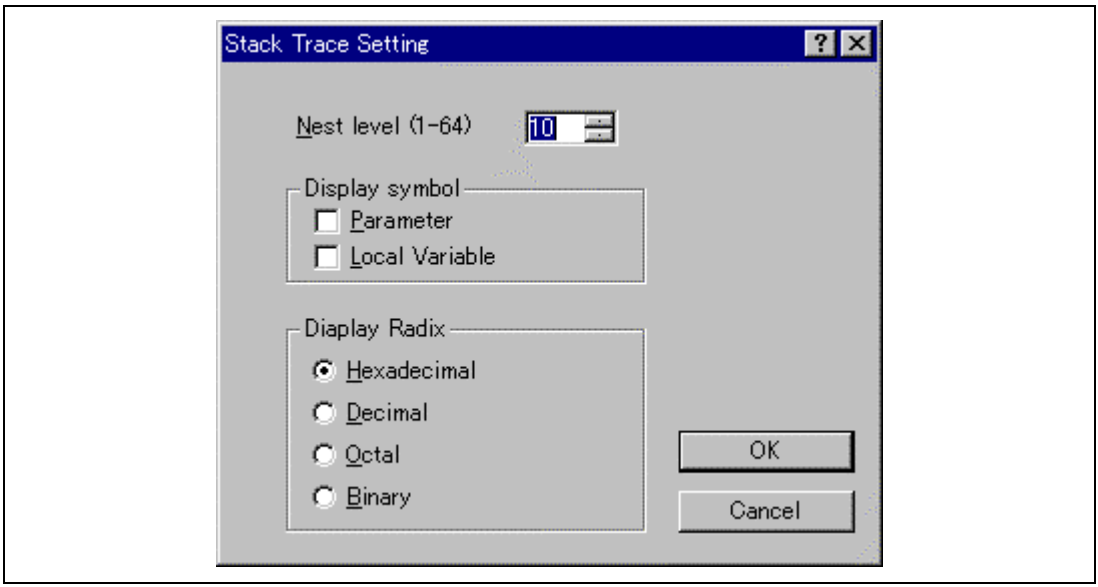


Figure 5.39 Stack Trace Setting Dialog Box

The following items are displayed.

[Nest level] Specifies the level of function call nesting to be displayed in the **Stack Trace** window.

[Display symbol] Specifies the symbol types to be displayed in addition to functions.

[Display Radix] Specifies the radix for displays in the **Stack Trace** window.

5.29.3 Copy

Copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

5.30 Profile Window (List Sheet)

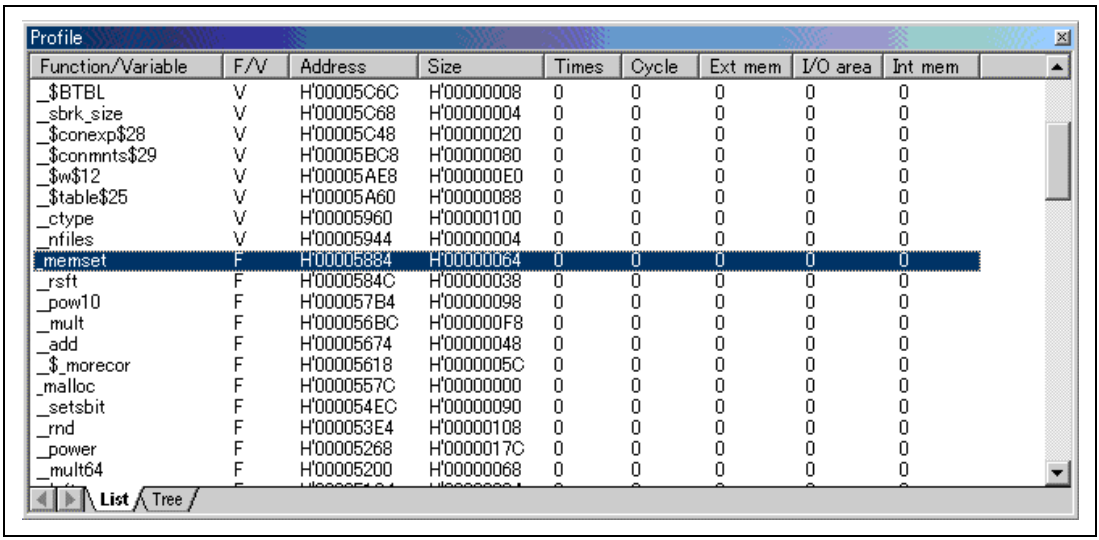


Figure 5.40 Profile Window (List Sheet)

This window displays the address and size of a function or a global variable, the number of times the function is called or the global variable is accessed, and profile data. The following profile data is displayed:

Times (the number of times a function is called or a global variable is accessed)
Cycle (the number of execution cycles)
Ext_mem (the number of external memory accesses)
I/O_area (the number of internal I/O accesses)
Int_mem (the number of internal memory accesses)

The number of execution cycles and cache misses are calculated by subtracting the total execution cycles or cache misses at a specific function call instruction execution from the total execution cycles or cache misses at a return instruction execution of a specific function.

When the column header is clicked, data are sorted in alphabetic or numeric ascending/descending order.

Double-clicking the **Function/Variable** or **Address** column displays the source program contents corresponding to the address in the line. Right-clicking on the mouse within the window displays a popup menu. For details on this popup menu, refer to section 5.31, Profile Window (Tree Sheet).

5.31 Profile Window (Tree Sheet)

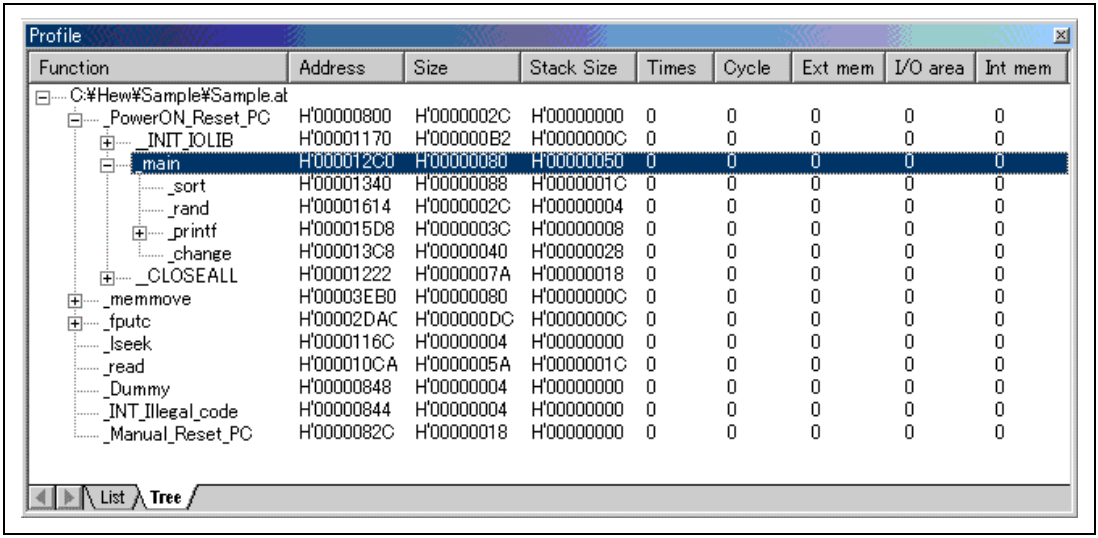


Figure 5.41 Profile Window (Tree Sheet)

This window displays the relation of function calls in a tree structure. Displayed contents are the address, size, stack size, number of function calls, and profile data. The stack size, number of function calls, and profile data are values when the function is called.

The following profile data is displayed:

Times (the number of times a function is called)
Cycle (the number of execution cycles)
Ext_mem (the number of external memory accesses)
I/O_area (the number of input/output area accesses)
Int_mem (the number of internal memory accesses)

The number of execution cycles and cache misses are calculated by subtracting the total execution cycles or cache misses at a specific function call instruction execution from the total execution cycles or cache misses at a return instruction execution of a specific function.

Note: Displayed stack size does not represent the actual size. Use it as a reference value when the function is called. If there is no stack information file (.sni extension) output from the optimizing linkage editor, the stack size is not displayed. For details of the stack information file, refer to the manual of the optimizing linkage editor.

Double-clicking a function in the Function column expands or reduces the tree structure display. The expansion or reduction is also provided by the “+” or “-“ key. Double-clicking the **Address** column displays the source program contents corresponding to the specific address.

Right-clicking on the mouse within the window displays a popup menu. Supported menu options are described in the following sections:

5.31.1 View Source

Displays the source program or disassembled memory contents for the address in the selected line.

5.31.2 View Profile-Chart

Displays the **Profile-Chart** window focused on the function in the specified line.

5.31.3 Enable Profiler

Toggles acquisition profile data. When profile data acquisition is active, a check mark is shown to the left of the menu text.

5.31.4 Not trace the function call

Stops tracing function calls while profile data is acquired. This menu is used when acquiring profile data of the program in which functions are called in a special way, such as task switching in the OS.

To display the relation of function calls in the **Tree** sheet of the **Profile** window, acquire profile data without selecting this menu. In addition, do not select this menu when optimizing the program by the optimizing linkage editor using the acquired profile information file.

5.31.5 Find...

Displays the **Find Text** dialog box to find a character string in the Function column. Search is started by inputting a character string to be found in the edit box and clicking **[Find Next]** or pressing ENTER.

5.31.6 Find Data...

Displays the **Find Data** dialog box.

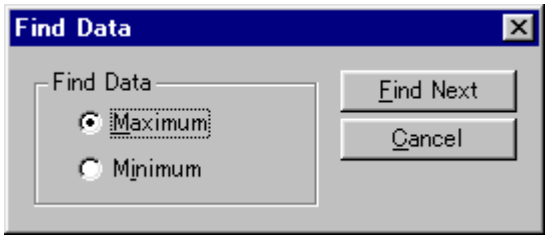


Figure 5.42 Find Data Dialog Box

By selecting the column to be searched in the **Column** combo box and the search type in the **Find Data** group and entering **[Find Next]** button or **ENTER** key, search is started. If the **[Find Next]** button or the **ENTER** key is input repeatedly, the second larger data (the second smaller data when the Minimum is specified) is searched for.

5.31.7 Clear Data

Clears the number of times functions are called and profile data. Data in the **Profile** window **List** sheet and the **Profile-Chart** window are also cleared.

5.31.8 Output Profile Information Files...

Displays the **Save Profile Information Files** dialog box. Profiling results are saved in a profile information file (.pro extension). The optimizing linkage editor optimizes user programs according to the profile information in this file. For details of the optimization using the profile information, refer to the manual of the optimizing linkage editor.

Note: If profile information has been acquired by selecting the Not trace the function call menu, the program cannot be optimized by the optimizing linkage editor.

5.31.9 Output Text File...

Displays the **Save Text of Profile Data** dialog box. Displayed contents are saved in a text file.

5.31.10 Setting

This menu has the following submenus (the menus available only in the **List** sheet are also included).

1. Show Functions/Variables

   Displays both functions and global variables in the Function/Variable column.
2. Show Functions

   Displays only functions in the Function/Variable column.
3. Show Variables

   Displays only global variables in the Function/Variable column.
4. Only Executed Functions

   Only displays the executed functions. If a stack information file (.sni extension) output from the optimizing linkage editor does not exist in the directory where the load module is located, only the executed functions are displayed even if this check box is not checked.
5. Include Data of Child Functions

   Sets whether or not to display information for a child function called in the function as profile data.

5.31.11 Properties...

This menu cannot be used in this simulator/debugger.

5.32 Profile Chart

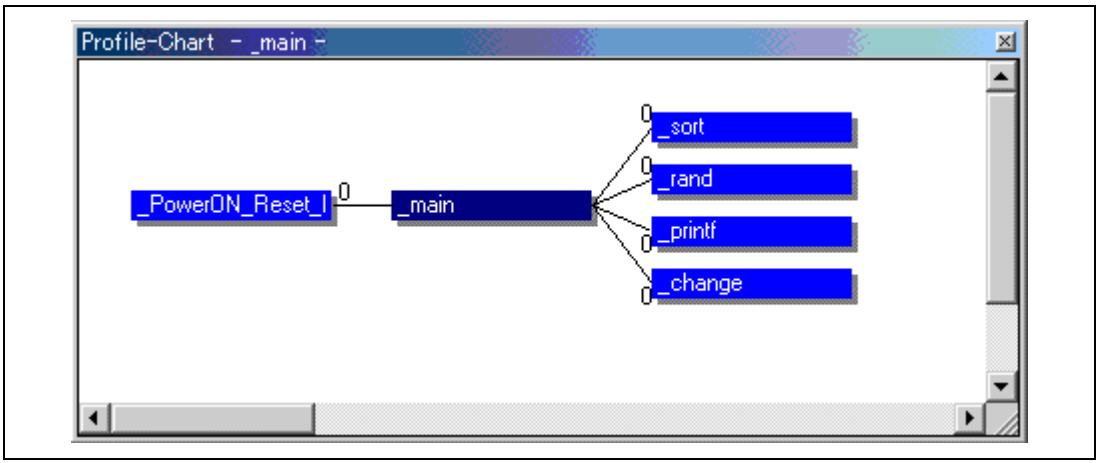


Figure 5.43 Profile-Chart Window

This window displays the relation of calls for a specific function. This window displays the calling relation for the function specified in the List sheet or Tree sheet in the **Profile** window. The specified function is displayed in the middle, the calling function on the left side, and the called function on the right side. Values beside the calling and called functions show the number of times the function has been called.

Right-clicking on the mouse within the window displays a popup menu. Supported menu options are described in the following.

5.32.1 View Source

Displays the source program or disassembled memory contents for the address of the function on which the cursor is placed when the right side button of the mouse is clicked. If the cursor is not placed on a function when the right side button is clicked, this menu option is displayed in gray characters.

5.32.2 View Profile-Chart

Displays the **Profile-Chart** window for the specific function on which the cursor is placed when the right side button of the mouse is clicked. If the cursor is not placed on a function when the right side button is clicked, this menu option is displayed in gray characters.

5.32.3 Enable Profiler

Toggles acquisition of profile data. When profile data acquisition is active, a check mark is shown to the left of the menu text.

5.32.4 Clear Data

Clears the number of times functions are called and profile data. Data in the **List** sheet and **Tree** sheet in the **Profile** window are also cleared.

5.32.5 Multiple View

If the **Profile-Chart** window is going to be opened when it has already been opened, selects whether another window is to be opened or the same window is to be used to display data. When a check mark is shown to the left side of the menu text, another window is opened.

5.32.6 Output Profile Information File...

Displays the **Save Profile Information File** dialog box. Profiling results are saved in a profile information file (.pro extension). The optimizing linkage editor optimizes user programs according to the profile information in this file. For details of the optimization using the profile information, refer to the manual of the optimizing linkage editor.

5.32.7 Expands Size

Expands spaces between each function. The “+” key can also be used to expand spaces.

5.32.8 Reduces Size

Reduces spaces between each function. The “-” key can also be used to reduce spaces.

5.33 Image View

Figure 5.44 Image View Window

Displays the contents of memory in an image. The method used to display the image can be specified in the **Image Properties** dialog box.

Double-clicking within the window displays information, in the **Pixel Information** dialog box, on the pixel on which the mouse pointer is located.

Right-clicking on the mouse within the window displays a popup menu. Supported menu options are described in the subsequent sections.

5.33.1 Auto Refresh

When this menu is checked, the contents of the window is automatically updated when the user program is executed.

5.33.2 Refresh Now

Updates the contents of the window.

5.33.3 Property...

Displays the **Image Properties** dialog box and specifies the format of the displayed image.

5.34 Image Properties Dialog Box

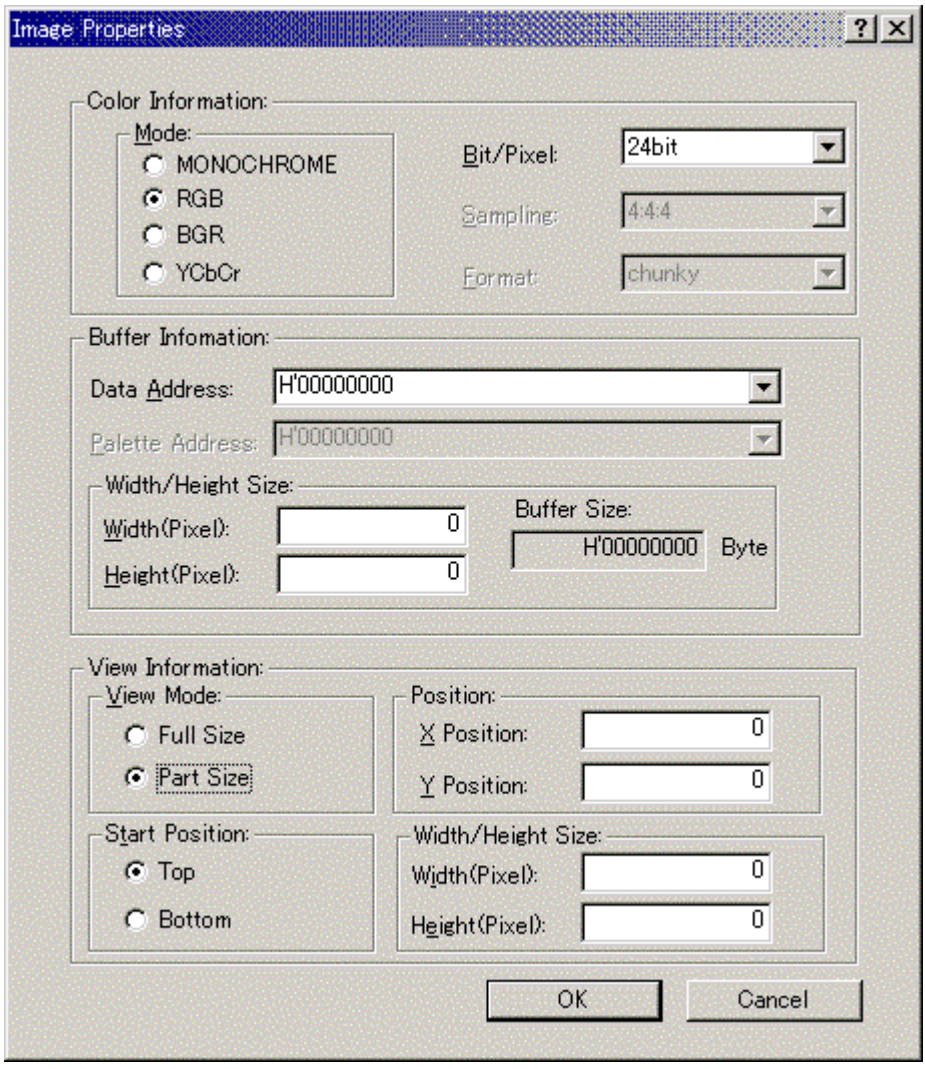


Figure 5.45 Image Properties Dialog Box

Specifies the method to display the **Image** window.

[Color Information] Specifies the color information of the image to be displayed.

[Mode] Specifies the format.

[MONOCHROME] Displays in black and white.

[RGB] Displayed in R (red), G (green), and B (blue)

[BGR] Displayed in B (blue), G (green), and R (red)

[YCbCr] Displayed by Y (brightness), Cb (color difference in blue), and Cr (color difference in red)

[Bit/Pixel] Specifies Bit/Pixel according to the selected **[Mode]**. (Valid when RGB/BGR is selected)

[Sampling] Specifies the format of sampling. (Valid when YCbCr is selected)

[Format] Specifies Chunky/planar. (Valid when YCbCr is selected)

[Buffer Information] Specifies the area to store data, size, and the address of the palette.

[Data Address] Specifies the start address of memory where image data is to be displayed. (Displayed in hexadecimal)

[Palette Address] Specifies the start address of memory of the color palette data. (Displayed in hexadecimal) (Valid when 8Bit is selected for RGB/BGR)

[Width/Height Size] Specifies the width and height of the image.

[Width (Pixel)] Specifies the width of the image. (When a prefix is omitted, the values are input and displayed in decimal.)

[Height (Pixel)] Specifies the height of the image. (When a prefix is omitted, the values are input and displayed in decimal.)

[Buffer Size] Displays the buffer size of the image from the width and height (Displayed in hexadecimal)

[View Information] Specifies the location, size, and the data start location of the part displayed among the entire image.

[View Mode] Specifies the entire/part to be displayed in the image.

[Full Size] Displays the entire image.

[Part Size] Displays part of the image.

[Start Position]

[Top] Displays data from the upper left.

[Bottom] Displays data from the lower left.

[Position] Specifies the start position of the image where part of the image is to be displayed. This is valid when **[Part Size]** is selected.

[X Position] Specifies the X axis of the start location. (When a prefix is omitted, the values are input and displayed in decimal.)

[Y Position] Specifies the Y axis of the start location. (When a prefix is omitted, the values are input and displayed in decimal.)

[Width/Height Size] Specifies the height and width of the image to be displayed partly.
[Width(Pixel)] Displays the width of display. (When a prefix is omitted, the values are input and displayed in decimal.)
[Height(Pixel)] Displays the height of display. (When a prefix is omitted, the values are input and displayed in decimal.)

5.35 Pixel Information

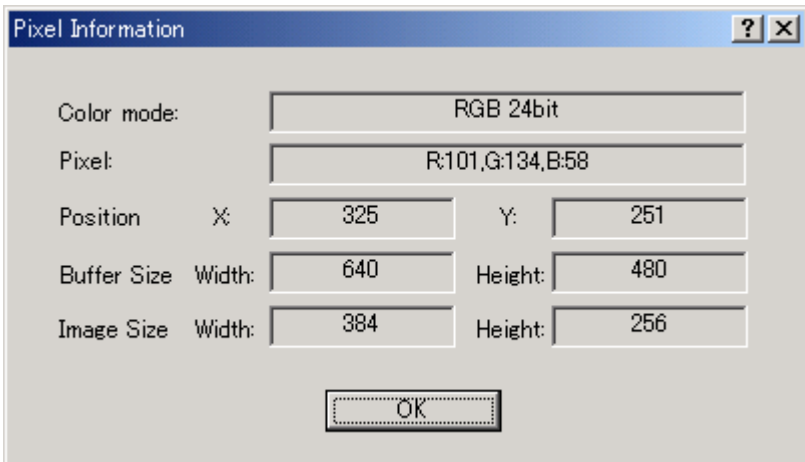


Figure 5.46 Pixel Information Window

The cursor location displays pixel information.

[Color Mode] Displays the format of the image.

[Pixel] Displays color information of the cursor location. (Displayed in decimal)

[Position] Displays the cursor location in X and Y axis. (Displayed in decimal)
[X] Displays the X axis of the cursor location.
[Y] Displays the Y axis of the cursor location.

[Buffer Size] Displays the buffer size. (Displayed in decimal)
[Width] Displays the buffer width.
[Height] Displays the buffer height.

[Image Size] Displays the width and height of the display. (Displayed in decimal)
[Width] Displays the width.
[Height] Displays the height.

5.36 Waveform Window

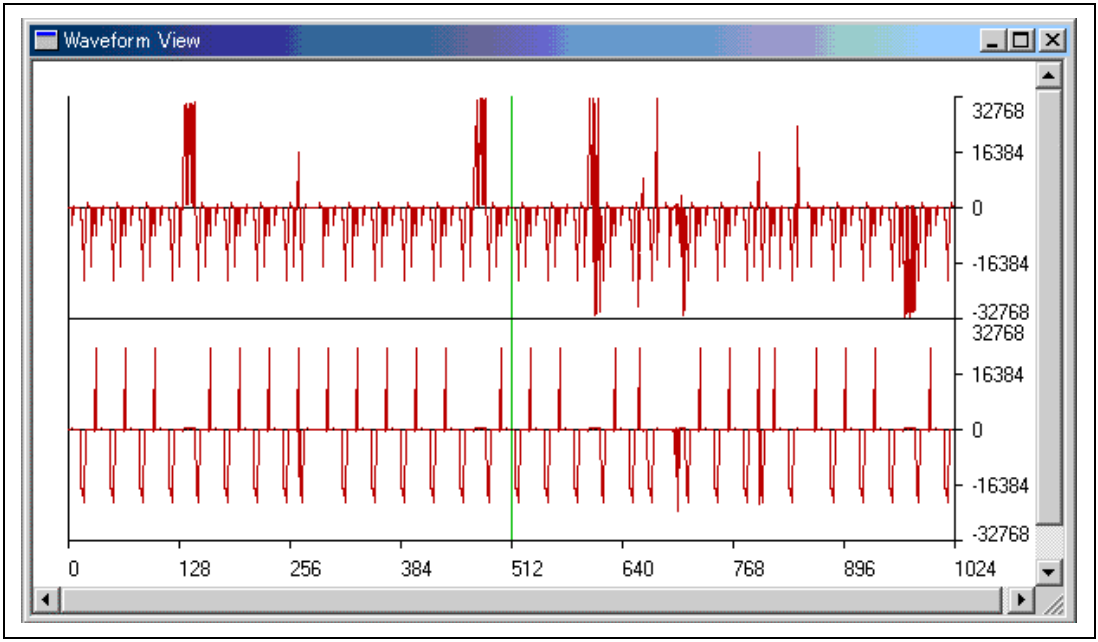


Figure 5.47 Waveform Window

Displays memory view in waveform. The X axis shows the number of sampling data and the Y axis shows the sampling value.

Clicking the right mouse button on the **Waveform View** window displays the popup menus. The menus include the following options.

5.36.1 Auto Refresh

Updates the display when instruction execution has stopped.

5.36.2 Refresh Now

Updates the display.

5.36.3 Zoom In

The horizontal axis is enlarged and displayed.

5.36.4 Zoom Out

The horizontal axis is reduced and displayed.

5.36.5 Reset Zoom

The size is returned to its original size.

5.36.6 Zoom Magnification

The zoom magnification can be selected from 2, 4, or 8 in the submenu.

5.36.7 Scale

The size of the X axis can be selected from 128, 256, or 512 Pixel from the submenu.

5.36.8 Clear Cursor

The cursor display is cleared.

5.36.9 Sample Information...

Displays the **Sample Information** dialog box. The sampling information is displayed.

5.36.10 Property...

Displays the **Waveform Property** dialog box. The waveform data format can be specified.

5.37 Waveform Properties Dialog Box

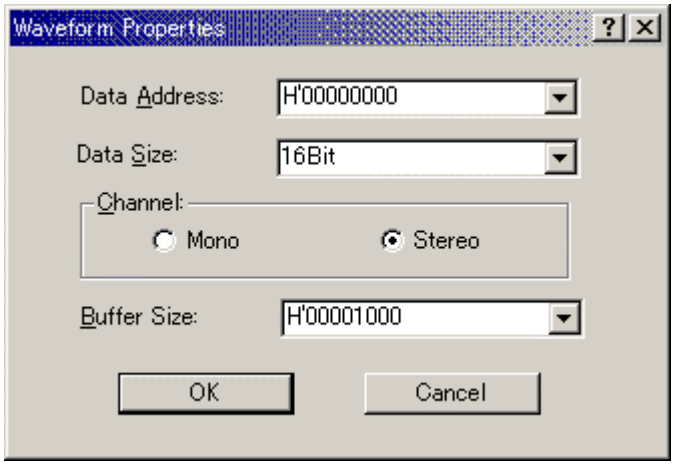


Figure 5.48 Waveform Properties Dialog Box

Specifies the waveform format. The following items can be specified.

[Data Address] Specifies the start address of data in memory. (Displayed in hexadecimal)

[Data Size] Selects 8Bit or 16Bit.

[Channel] Specifies Mono or Stereo.

[Buffer Size] Specifies the buffer size of data. (Displayed in hexadecimal)

5.38 Sample Information Dialog Box

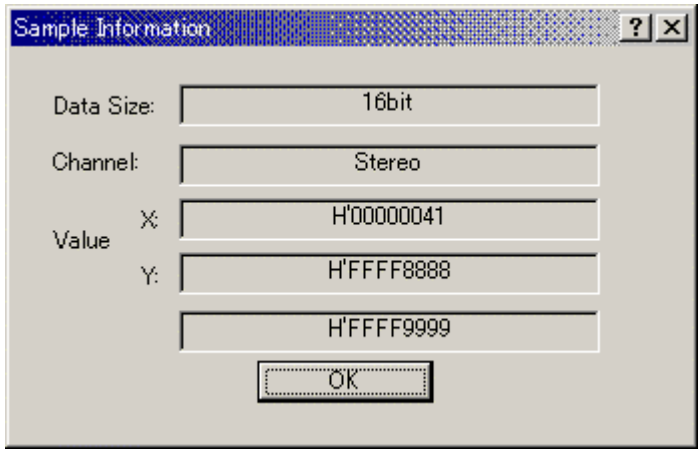


Figure 5.49 Sample Information Dialog Box

Displays the sampling information of the cursor location in the **Waveform View** window. The following information is displayed.

[Data Size] Displays 8bit or 16bit.

[Channel] Displays the data channel.

[Value] [X] Displays the X axis of cursor location.

[Y] Displays the Y axis of cursor location (displays Y axes for both the upper and lower plots when Stereo is selected).

5.39 Coverage

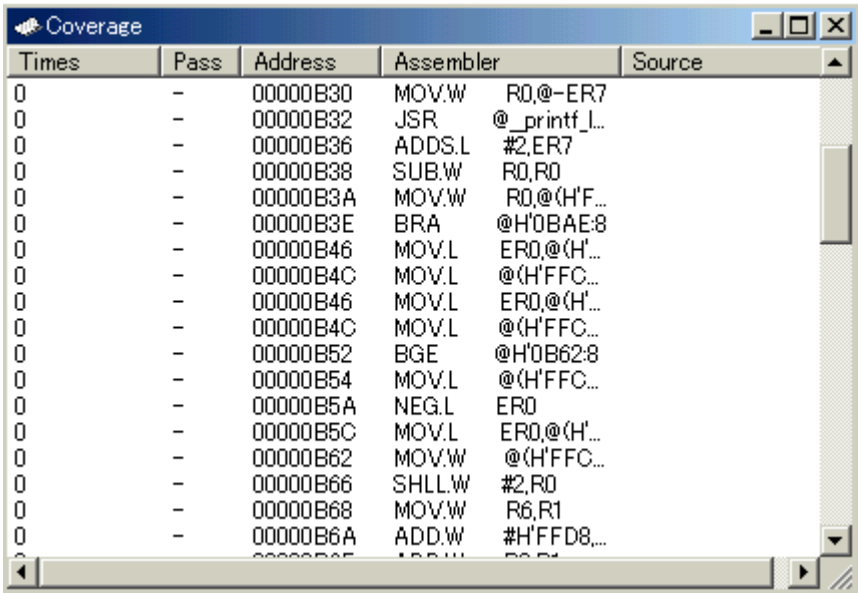


Figure 5.50 Coverage Window

Displays the instruction execution information in C/C++ and assembly-language level. Note that the conditions to acquire instruction execution information can be set through the **Open Coverage** dialog box or the **Coverage Range** dialog box. When the **Coverage** window is closed, the settings of the acquired instruction execution information and the conditions to acquire information will be cleared.

The items to be displayed are as follows:

[Times] Number of times instruction was executed

[Pass] Conditions to execute condition branch instructions
True: Branches because the condition is satisfied
False: Does not branch because the condition is not satisfied

[Address] Instruction Address

[Assembler] Disassembled and then displayed.

[Source] C/C++ or assembler source

Clicking the right mouse button on the **Coverage** window displays the popup menus. The menus include the following options.

5.39.1 View Source

Displays the **Source** window corresponding to the cursor location on the **Coverage** window.

5.39.2 Go to Address...

Modifies the address displayed in the **Coverage** window.

5.39.3 Set Range...

Displays the **Coverage Range** dialog box, allowing the user to specify the conditions to acquire instruction execution information.

5.39.4 Enable Coverage

Sets whether to enable or disable the acquisition of instruction execution information.

5.39.5 Clear Data...

Clears the acquired instruction execution information.

5.39.6 Save Data...

Launches the **Save Data** dialog box, allowing the user to save the coverage information in a file.

5.39.7 Load Data...

Launches the **Load Data** dialog box, allowing the user to load the coverage information from a file.

5.39.8 Refresh

Displays the latest instruction execution information.

5.39.9 Lock Refresh

Lock or unlock the refresh of coverage information. Note that when [Lock Refresh] is [On], window updates only the Times and Pass column after program execution.

5.40 Open Coverage Dialog Box

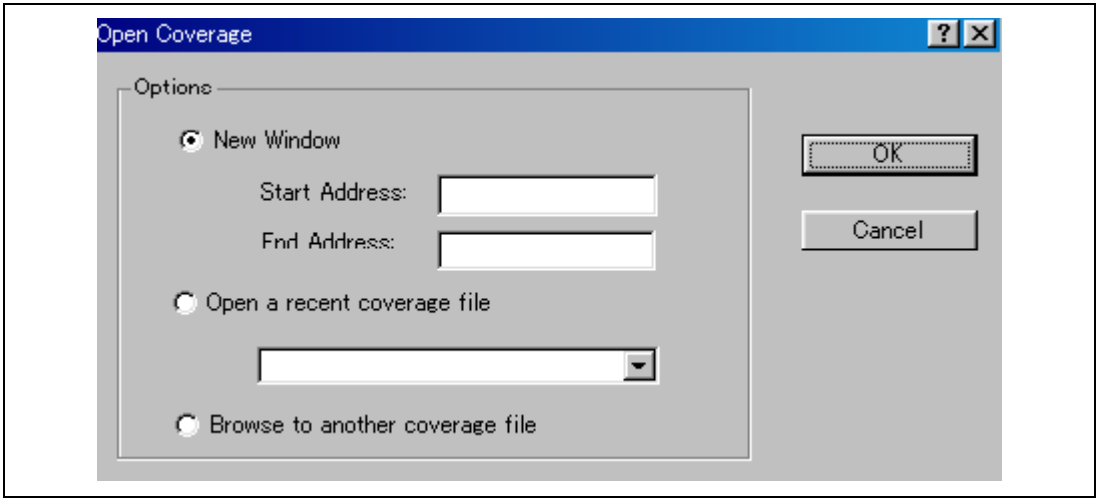


Figure 5.51 Open Coverage Dialog Box

Clicking the coverage icon opens the **Open Coverage** dialog box.

[New Window], **[Open a recent coverage file]**, or **[Browse to another coverage file]** can be selected. When **[New Window]** is selected, the start and end addresses of the coverage information range to be displayed must be specified as follows:

[Start Address] Start address of coverage information display
(When a prefix is omitted, the values is input in hexadecimal.)

[End Address] End address of coverage information display
(When a prefix is omitted, the value is input in hexadecimal.)

When **[Open a recent coverage file]** is selected, up to four recent files that have been saved are displayed.

When **[Browse to another coverage file]** is selected, a file open dialog box will appear to prompt the user to select a coverage information file.

5.41 Go To Address Dialog Box

Figure 5.52 Go To Address Dialog Box

Modifies the address displayed in the **Coverage** window.

5.42 Coverage Range Dialog Box

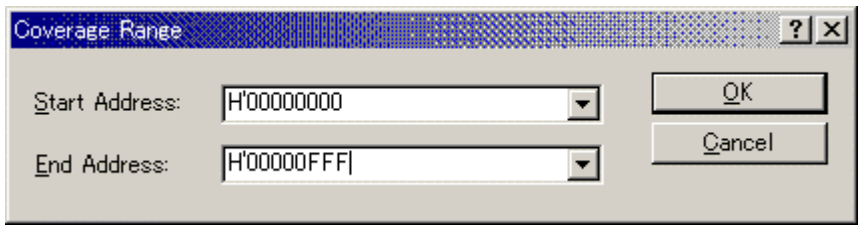


Figure 5.53 Coverage Range Dialog Box

Specifies the condition to acquire instruction execution information. The following items can be specified.

[Start Address] Start address (When a prefix is omitted, the values are input in hexadecimal.)

[End Address] End address (When a prefix is omitted, the values are input in hexadecimal.)

5.43 Save Data Dialog Box

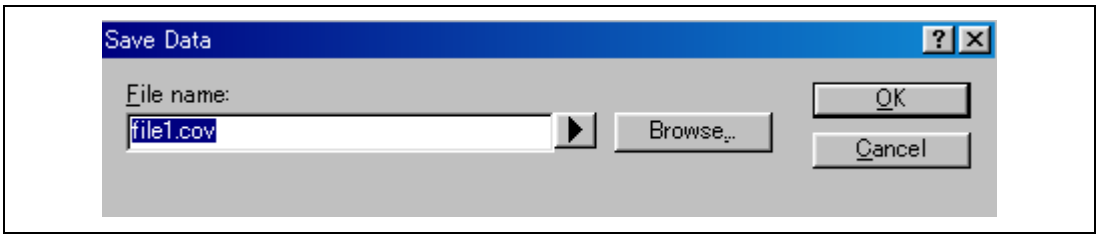


Figure 5.54 Save Data Dialog Box

Specifies the location and name of a coverage information file to be saved. The placeholder or the browse button can be used.

If a file name extension is omitted, .COV is automatically added. If a file name extension other than .COV or .TXT is specified, an error message will be displayed.

5.44 Load Data Dialog Box

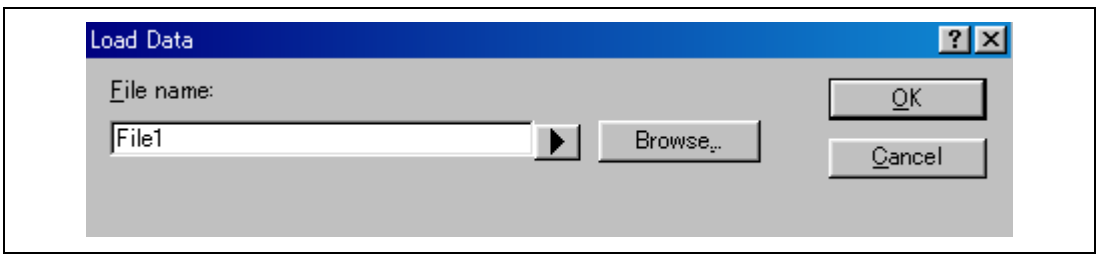


Figure 5.55 Load Data Dialog Box

Specifies the location and name of a coverage information file to be loaded. The placeholder or the browse button can be used.

Only .COV files can be loaded. If a file name extension other than .COV is specified, an error message will be displayed.

5.45 Confirmation Request Dialog Box

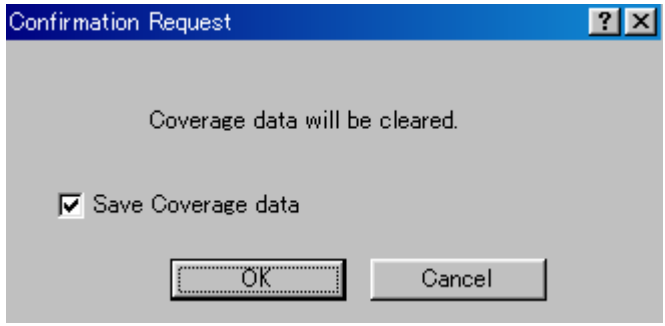


Figure 5.56 Confirmation Request Dialog Box

When **[Clear Data]** or **[Set Range...]** is clicked or when an attempt is made to close coverage window, the **Confirmation Request** dialog box will appear.

5.46 Save Coverage Data Dialog Box

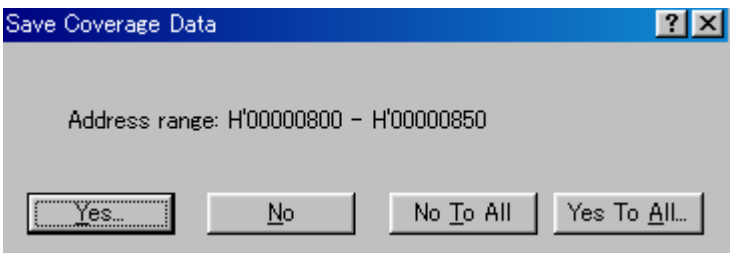


Figure 5.57 Save Coverage Data Dialog Box

When **[Save Session]** is selected from the **[File]** menu, the **Save Coverage Data** dialog box will appear, allowing the user to save the coverage window data in separate files or a single file.

When multiple coverage windows are open, the **Save Coverage Data** dialog box will appear for each open coverage window.

Clicking the **[Not To All]** button close the dialog box without saves the data.

Clicking the **[Yes To All]** button saves the data of all coverage windows in a single file.

Table 6.1 lists the commands.

Table 6.1 Simulator/Debugger Commands

No.	Command Name	Abbreviation	Function
1	!	-	Comment
2	ANALYSIS	AN	Enables or disables performance analysis
3	ANALYSIS_RANGE	AR	Sets or displays performance analysis functions
4	ANALYSIS_RANGE_ DELETE	AD	Deletes a performance analysis range
5	ASSEMBLE	AS	Assembles instructions into memory
6	ASSERT	-	Checks if an expression is true or false
7	BREAKPOINT	BP	Sets a breakpoint at an instruction address
8	BREAK_ACCESS	BA	Specifies a memory range access as a break condition
9	BREAK_CLEAR	BC	Deletes breakpoints
10	BREAK_CYCLE	BCY	Specifies a cycle as a break condition
11	BREAK_DATA	BD	Specifies a memory data value as a break condition
12	BREAK_DISPLAY	BI	Displays a list of breakpoints
13	BREAK_ENABLE	BE	Enables or disables a breakpoint
14	BREAK_REGISTER	BR	Specifies a register data as a break condition
15	BREAK_SEQUENCE	BS	Sets sequential breakpoints
16	CHANGE_CONFIGURA TION	CC	Sets the current configuration
17	CHANGE_PROJECT	CP	Sets the current project
18	COVERAGE	CV	Enables or disables coverage measurement
19	COVERAGE_DISPLAY	CVD	Displays coverage information
20	COVERAGE_LOAD	CVL	Loads coverage information
21	COVERAGE_RANGE	CVR	Sets a coverage range
22	COVERAGE_SAVE	CVS	Saves coverage information
23	DEFAULT_OBJECT_ FORMAT	DO	Sets the default object (program) format
24	DISASSEMBLE	DA	Disassembles memory contents
25	ERASE	ER	Clears the **Command Line** window
26	EVALUATE	EV	Evaluates an expression
27	FILE_LOAD	FL	Loads an object (program) file

Table 6.1 Simulator/Debugger Commands (cont)

No.	Command Name	Abbreviation	Function
28	FILE_SAVE	FS	Saves memory to a file
29	FILE_VERIFY	FV	Verifies file contents against memory
30	GO	GO	Executes user program
31	GO_RESET	GR	Executes user program from reset vector
32	GO_TILL	GT	Executes user program until temporary breakpoint
33	HALT	HA	Halts the user program
34	INITIALIZE	IN	Initializes the debugging platform
35	LOG	LO	Controls command output logging
36	MAP_DISPLAY	MA	Displays memory mapping
37	MAP_SET	MS	Allocates a memory area
38	MEMORY_DISPLAY	MD	Displays memory contents
39	MEMORY_EDIT	ME	Modifies memory contents
40	MEMORY_FILL	MF	Fills a memory area
41	MEMORY_MOVE	MV	Moves a block of memory
42	MEMORY_TEST	MT	Tests a block of memory
43	OPEN_WORKSPACE	OW	Opens a workspace
44	PROFILE	PR	Enables or disables profile
45	PROFILE_DISPLAY	PD	Displays profile information
46	PROFILE_SAVE	PS	Saves the profile information to file
47	QUIT	QU	Exits HEW
48	RADIX	RA	Sets default input radix
49	REGISTER_DISPLAY	RD	Displays CPU register values
50	REGISTER_SET	RS	Changes CPU register contents
51	RESET	RE	Resets CPU
52	RESPONSE	RS	Sets a window refresh area
53	SLEEP	-	Delays command execution
54	STEP	ST	Steps program (by instructions or source lines)
55	STEP_MODE	SM	Selects the step mode
56	STEP_OUT	SP	Steps out of the current function
57	STEP_OVER	SO	Steps program, not stepping into functions
58	STEP_RATE	SR	Sets or displays rate of stepping
59	SUBMIT	SU	Executes a command file
60	SYMBOL_ADD	SA	Defines a symbol

Table 6.1 Simulator/Debugger Commands (cont)

No.	Command Name	Abbreviation	Function
61	SYMBOL_CLEAR	SC	Deletes a symbol
62	SYMBOL_LOAD	SL	Loads a symbol information file
63	SYMBOL_SAVE	SS	Saves a symbol information file
64	SYMBOL_VIEW	SV	Displays symbols
65	TCL	-	Enables or disables the TCL
66	TRACE	TR	Displays trace information
67	TRACE_ACQUISITION	TA	Enables or disables trace information acquisition
68	TRACE_SAVE	TV	Outputs trace information into a file
69	TRACE_STATISTIC	TST	Analyzes statistic information

The following describes the syntax of each command.

6.1 !(COMMENT)

Abbreviation: none

Description:

Allows a comment to be entered, useful for documenting log files.

Syntax:

! <text>

Parameter	Type	Description
<text>	Text	Output text

Example:

! Start of test routine	Outputs comment 'Start of test routine' into the **Command Line** window (and to the log file, if logging is active).

6.2 ANALYSIS

Abbreviation: AN

Description:

Enables/disables performance analysis. Counts are not automatically reset before running.

Syntax:

an [<state>]

Parameter	Type	Description
None		Displays the performance analysis state
<state>	Keyword	Enables/disables performance analysis
	Enable	Enables performance analysis
	Disable	Disables performance analysis
	Reset	Resets performance analysis counts

Examples:

ANALYSIS	Displays performance analysis state.
AN enable	Enables performance analysis.
AN disable	Disables performance analysis.
AN reset	Resets performance analysis counts.

6.3 ANALYSIS_RANGE

Abbreviation: AR

Description:

Sets a function for which the performance analysis is provided, or displays a function for which the performance analysis is provided without parameters.

Syntax:

ar [<function name>]

Parameter	Type	Description
None		Displays all functions for which the performance analysis is provided
<function name>	Character string	Name of function for which the performance analysis is provided

Examples:

ANALYSIS_RANGE sort	Provides the performance analysis for the function sort.
AR	Displays the function for which the performance analysis is provided.

6.4 ANALYSIS_RANGE_DELETE

Abbreviation: AD

Description:

Deletes the specified function, or all functions if no parameters are specified (it does **not** ask for confirmation).

Syntax:

ad [<index>]

Parameter	Type	Description
None		Deletes all functions
<index>	Numeric	Index number of function to delete

Examples:

ANALYSIS_RANGE_DELETE 6	Deletes the function with index number 6.
AD	Deletes all functions.

6.5 ASSEMBLE

Abbreviation: AS

Description:

Assembles mnemonics and writes them into memory. In assembly mode, '.' exits, '^' steps back a byte, the ENTER key steps forward a byte.

Syntax:

as <address>

Parameter	Type	Description
<address>	Numeric	Address at which to start assembling

Example:

AS H’1000	Starts assembling from H’1000.

6.6 ASSERT

Abbreviation: none

Description:

Checks if an expression is true or false. It can be used to terminate the batch file when the expression is false. If the expression is false, an error is returned. This command can be used to write test harnesses for subroutines.

Syntax:

assert <expression>

Parameter	Type	Description
<expression>	Expression	Expression to be checked

Example:

ASSERT #R0 == 0x100	Returns an error if R0 does not contain 0x100.

6.7 BREAKPOINT

Abbreviation: BP

Description:

Specifies a breakpoint at the address where the instruction is written.

Syntax:

bp <address> [<count>] [<Action>]

Parameter	Type	Description
<address>	Numeric	The address of a breakpoint
<count>	Numeric	The number of times the instruction at the specified address is to be fetched (1 to 16383, default = 1).
<Action>	Keyword	Action taken when the conditions are satisfied (optional, default = Stop)
	Stop (P)	Halts the execution of the user program
	Input (I)	Inputs (saves) data to a file
	Output (O)	Outputs (reads) data from a file
	Interrupt (T)	Initiates a pseudo-interrupt

Format:

The method of defining each Action are as follows.

Stop

Input <filename> <addr> <size> <count>

Parameter	Type	Description
<filename>	Character string	The name of the file from which data is input
<addr>	Numeric	Address to which the data is read.
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H'01 to H'FFFFFFFF)

Output <filename> <addr> <size> <count> [<option>]

Parameter	Type	Description
<filename>	Character string	The name of the file to which data is saved
<addr>	Numeric	Address from which data is output
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H'01 to H'FFFFFFFF)
<option>	Keyword	Specifies a new file or appends to an existing file. (optional, makes a new file when abbreviated.)
	A	Adds the data to the existing file.

Interrupt <interrupt type1> [<priority>]

Parameter	Type	Description
<interrupt type1>	Numeric	Type of interrupt Interrupt vector number (0 to H'FF)
<priority>	Numeric	Interrupt priority (optional, default = 0) 0 to 17

Examples:

BREAKPOINT 0 2	A break occurs when an attempt is made to execute the instruction at address H'0 for the second time.
BP C0 Input in.dat 100 2 8	Eight two-byte data fields are written from file “in.dat” to H’100 when an attempt is made to execute the instruction at address H'C0.

6.8 BREAK_ACCESS

Abbreviation: BA

Description:

Specifies a memory range as a break condition

Syntax:

ba <start address> [<end address>] [<mode>] [<Action>]

Parameter	Type	Description
<start address>	Numeric	The start address of a breakpoint
<end address>	Numeric	The end address of a breakpoint (optional, default = <start address>)
<mode>	Keyword	Access type (optional, default = RW).
	R	A break occurs when the specified range is read.
	W	A break occurs when the specified range is written to.
	RW	A break occurs when the specified range is read or written to.
<Action>	Keyword	Action taken when the conditions are satisfied (optional, default = Stop)
	Stop (P)	Halts the execution of the user program
	Input (I)	Inputs (saves) data to a file
	Output (O)	Outputs (reads) data from a file
	Interrupt (T)	Initiates a pseudo-interrupt

Format:

The method of defining each Action are as follows.

Stop

Input <filename> <addr> <size> <count>

Parameter	Type	Description
<filename>	Character string	The name of the file from which data is input
<addr>	Numeric	Address to which the data is read.
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H’01 to H’FFFFFFFF)

Output <filename> <addr> <size> <count> [<option>]

Parameter	Type	Description
<filename>	Character string	The name of the file to which data is saved
<addr>	Numeric	Address from which data is output
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H’01 to H’FFFFFFFF)
<option>	Keyword	Specifies a new file or appends to an existing file. (optional, makes a new file when abbreviated.)
	A	Adds the data to the existing file.

Interrupt <interrupt type1> [<priority>]

Parameter	Type	Description
<interrupt type1>	Numeric	Type of interrupt Interrupt vector number (0 to H’FF)
<priority>	Numeric	Interrupt priority (optional, default = 0) 0 to 17

Examples:

BREAK_ACCESS 0 1000 W	A break occurs when the specified range from address H'0 to address H'1000 is written to.
BA FFFF	A break occurs when address H'FFFF is accessed.

6.9 BREAK_CLEAR

Abbreviation: BC

Description:

Deletes breakpoints.

Syntax:

bc <index>

Parameter	Type	Description
<index>	Numeric	Index of the breakpoint to be canceled. If the index is omitted, all breakpoints are deleted.

Examples:

BREAK_CLEAR 0	The first breakpoint is deleted.
BC	All breakpoints are deleted.

6.10 BREAK_CYCLE

Abbreviation: BCY

Description:

Specifies the number of cycles as a break condition.

Syntax:

by <cycle> [<count>] [<Action>]

Parameter	Type	Description
<cycle>	Numeric	The condition matching the number of cycles <cycle>×n.
<count>	Keyword	The condition satisfying the number of times. (optional, default =ALL)
	All	Break condition is satisfied every time the condition is matched.
	Numeric	1 to H'FFFF Break condition is satisfied only when it matches the specified number of times.
<Action>	Keyword	Action taken when the conditions are satisfied (optional, default = Stop)
	Stop (P)	Halts the execution of the user program
	Input (I)	Inputs(saves) data to a file
	Output (O)	Outputs(reads) data from a file
	Interrupt (T)	Initiates a pseudo-interrupt

Format:

The method of defining each Action are as follows.

Stop

Input <filename> <addr> <size> <count>

Parameter	Type	Description
<filename>	Character string	The name of the file from which data is input
<addr>	Numeric	Address to which the data is read.
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H'01 to H'FFFFFFFF)

Output <filename> <addr> <size> <count> [<option>]

Parameter	Type	Description
<filename>	Character string	The name of the file to which data is saved
<addr>	Numeric	Address from which data is output
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H'01 to H'FFFFFFFF)
<option>	Keyword	Specifies a new file or appends to an existing file. (optional, makes a new file when abbreviated.)
	A	Adds the data to the existing file.

Interrupt <interrupt type1> [<priority>]

Parameter	Type	Description
<interrupt type1>	Numeric	Type of interrupt Interrupt vector number (0 to H'FF)
<priority>	Numeric	Interrupt priority (optional, default = 0) 0 to 17

Examples:

BREAK_CYCLE 1000 20	Specifies breaks to occur H'20 times in every H'1000 cycles.
BCY 5000	Specifies a break to occur in every H'5000 cycles.

6.11 BREAK_DATA

Abbreviation: BD

Description:

Specifies a memory data value as a break condition.

Syntax:

bd <address> <data> [<size>] [<option>] [<Action>]

Parameter	Type	Description
<address>	Numeric	The address where the break condition is checked.
<data>	Numeric	Access data
<size>	Keyword	Size (optional, default = B).
	B	Byte size
	W	Word size
	L	Longword size
	S	Single-precision floating-point size
	D	Double-precision floating-point size
<option>	Keyword	Match or mismatch of data. The default is EQ.
	EQ	A break occurs when the data matches the specified value.
	NE	A break occurs when the data does not match the specified value.
<Action>	Keyword	Action taken when the conditions are satisfied (optional, default = Stop)
	Stop (P)	Halts the execution of the user program
	Input (I)	Inputs(saves) data to a file
	Output (O)	Outputs(reads) data from a file
	Interrupt (T)	Initiates a pseudo-interrupt

Format:

The method of defining each Action are as follows.

Stop

Input <filename> <addr> <size> <count>

Parameter	Type	Description
<filename>	Character string	The name of the file from which data is input
<addr>	Numeric	Address to which the data is read.
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H’01 to H’FFFFFFFF)

Output <filename> <addr> <size> <count> [<option>]

Parameter	Type	Description
<filename>	Character string	The name of the file to which data is saved
<addr>	Numeric	Address from which data is output
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H'01 to H'FFFFFFFF)
<option>	Keyword	Specifies a new file or appends to an existing file. (optional, makes a new file when abbreviated.)
	A	Adds the data to the existing file.

Interrupt <interrupt type1> [<priority>]

Parameter	Type	Description
<interrupt type1>	Numeric	Type of interrupt Interrupt vector number (0 to H'FF)
<priority>	Numeric	Interrupt priority (optional, default = 0) 0 to 17

Examples:

BREAK_DATA 0 100 L EQ	A break occurs when H'100 is written to memory address H'0 in longword.
BD C0 FF B NE	A break occurs when a value other than H'FF is written to memory address H'C0 in byte.
BD 4000 10	A break occurs when H'10 is written to memory address H'4000 in byte.

6.12 BREAK_DISPLAY

Abbreviation: BI

Description:

Displays a list of breakpoints.

Syntax:

bi

Parameter	Type	Description
None		Displays a list of breakpoints

Examples:

BREAK_DISPLAY	A list of breakpoints is displayed.
BI	A list of breakpoints is displayed.

6.13 BREAK_ENABLE

Abbreviation: BE

Description:

Enables or disables a breakpoint.

Syntax:

be <flag> [<index>]

Parameter	Type	Description
<flag>	Keyword	Enabling or disabling of a breakpoint
	E	Enable
	D	Disable
<index>	Numeric	Index of the breakpoint to be canceled. If the index is omitted, all breakpoints are deleted.

Examples:

BREAK_ENABLE D 0	The first breakpoint is disabled.
BE E	All breakpoints are enabled.

6.14 BREAK_REGISTER

Abbreviation: BR

Description:

Specifies a register data as a break condition

Syntax:

br <register name> [<data> <size>] [<option>] [<Action>]

Parameter	Type	Description
<register>	Character string	Register name.
<data>	Numeric	Access data.
<size>	Keyword	Access size. If no size is specified, the size of the specified register is assumed. Note that when data is specified, the size must not be omitted.
	B	Byte size
	W	Word size
	L	Longword size
	S	Single-precision floating-point size
	D	Double-precision floating-point size
<option>	Keyword	Match or mismatch of data. The default is EQ.
	EQ	A break occurs when the data matches the specified value.
	NE	A break occurs when the data does not match the specified value.
<Action>	Keyword	Action taken when the conditions are satisfied (optional, default = Stop)
	Stop (P)	Halts the execution of the user program
	Input (I)	Inputs(saves) data to a file
	Output (O)	Outputs(reads) data from a file
	Interrupt (T)	Initiates a pseudo-interrupt

Format:

The method of defining each Action are as follows.

Stop

Input <filename> <addr> <size> <count>

Parameter	Type	Description
<filename>	Character string	The name of the file from which data is input
<addr>	Numeric	Address to which the data is read.
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H’01 to H’FFFFFFFF)

Output <filename> <addr> <size> <count> [<option>]

Parameter	Type	Description
<filename>	Character string	The name of the file to which data is saved
<addr>	Numeric	Address from which data is output
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H’01 to H’FFFFFFFF)
<option>	Keyword	Specifies a new file or appends to an existing file. (optional, makes a new file when abbreviated.)
	A	Adds the data to the existing file.

Interrupt <interrupt type1> [<priority>]

Parameter	Type	Description
<interrupt type1>	Numeric	Type of interrupt Interrupt vector number (0 to H’FF)
<priority>	Numeric	Interrupt priority (optional, default = 0) 0 to 17

Examples:

BREAK_REGISTER R0 FFFF W EQ	A break occurs when the lower two bytes of the R0 register change to H'FFFF.
BR R10	A break occurs when the R10 register is written to.

6.15 BREAK_SEQUENCE

Abbreviation: BS

Description:

Sets sequential breakpoints

Syntax:

bs <address1> [<address2> [<address 3> [...]]] [<Action>]

Parameter	Type	Description
<address1> - <address8>	Numeric	Addresses of sequential breakpoints. Up to eight addresses can be specified.
<Action>	Keyword	Action taken when the conditions are satisfied (optional, default = Stop)
	Stop (P)	Halts the execution of the user program
	Input (I)	Inputs(saves) data to a file
	Output (O)	Outputs(reads) data from a file
	Interrupt (T)	Initiates a pseudo-interrupt

Format:

The method of defining each Action are as follows.

Stop

Input <filename> <addr> <size> <count>

Parameter	Type	Description
<filename>	Character string	The name of the file from which data is input
<addr>	Numeric	Address to which the data is read.
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H’01 to H’FFFFFFFF)

Output <filename> <addr> <size> <count> [<option>]

Parameter	Type	Description
<filename>	Character string	The name of the file to which data is saved
<addr>	Numeric	Address from which data is output
<size>	Numeric	Size per data packet (1/2/4/8)
<count>	Numeric	Number of data packets (H'01 to H'FFFFFFFF)
<option>	Keyword	Specifies a new file or appends to an existing file. (optional, makes a new file when abbreviated.)
	A	Adds the data to the existing file.

Interrupt <interrupt type1> [<priority>]

Parameter	Type	Description
<interrupt type1>	Numeric	Type of interrupt Interrupt vector number (0 to H'FF)
<priority>	Numeric	Interrupt priority (optional, default = 0) 0 to 17

Examples:

BREAK_SEQUENCE 1000 2000	A break occurs when addresses H'1000 and H'2000 are passed in this order.
BS 1000	A break occurs when address H'1000 is executed.

6.16 CHANGE_CONFIGURATION

Abbreviation: CC

Description:

Sets the current configuration.

Syntax:

cc <config name>

Parameter	Type	Description
<config name>	Character string	Configuration name

Example:

CC Debug	Sets the current configuration to Debug.

6.17 CHANGE_PROJECT

Abbreviation: CP

Description:

Sets the current project.

Syntax:

cp <project name>

Parameter	Type	Description
<project name>	Character string	Project name

Example:

CP PROJ2	Sets the current project to PROJ2.

6.18 COVERAGE

Abbreviation: CV

Description:

Enables or disables the coverage range measurement or resets coverage information.

Syntax:

co [<state>]

Parameter	Type	Description
none		Displays coverage state.
<state>	enable	Enables coverage measurement.
	disable	Disables coverage measurement.
	reset	Resets result of coverage measurement.

Examples:

COVERAGE	Displays coverage state.
CV enable	Enables coverage measurement.
CV r	Resets result of coverage measurement.

6.19 COVERAGE_DISPLAY

Abbreviation: CVD

Description:

Displays coverage information.

Syntax:

cvd

Parameter	Type	Description
none		Displays coverage information.

Example:

COVERAGE_DISPLAY	Displays coverage information.

6.20 COVERAGE_LOAD

Abbreviation: CVL

Description:

Loads the coverage information from a .COV file.
If a wrong file format is specified or the specified file is not found, a warning message will be displayed.

Syntax:

cvl <filename>

Parameter	Type	Description
filename	Character string	File name

Examples:

COVERAGE_LOAD TEST	Loads the coverage information from the TEST.COV file.
CLV COVERAGE.COV	Loads the coverage information from the COVERAGE.COV file.

6.21 COVERAGE_RANGE

Abbreviation: CVR

Description:

Sets the coverage range or displays the range of coverage measurement without parameters.

Syntax:

cvr [<start> <end>]

Parameter	Type	Description
none		Displays the coverage measurement range.
<start>	Numeric	Start address of the coverage measurement range.
<end>	Numeric	End address of the coverage measurement range.

Examples:

COVERAGE_RANGE H'1000 H'10FF	Measures the coverage of addresses between H'1000 and H'10FF.
CVR	Displays the range of coverage measurement.

6.22 COVERAGE_SAVE

Abbreviation: CVS

Description:

Saves the coverage information in a .COV file.
If a wrong file extension is specified, an error message will be displayed.

Syntax:

cvs <filename>

Parameter	Type	Description
filename	Character string	File name

Examples:

COVERAGE_SAVE TEST	Saves the coverage information in the TEST.COV file.
CVS COVERAGE.COV	Saves the coverage information in the COVERAGE.COV file.

6.23 DEFAULT_OBJECT_FORMAT

Abbreviation: DO

Description:

Sets the default format for loading object (program) files. The format set with this command is only valid when the format specification is omitted from the FILE_LOAD command.

Syntax:

do [<format>]

Parameter	Type	Description
none		Displays the default format settings.
<format>	Keyword	Object format
	Binary	Binary type
	Elf/Dwarf2	Elf/Dwarf2 type
	IntelHex	Intel-Hex type
	S-Record	S type

Example:

DEFAULT_OBJECT_FORMAT	Displays the default format settings.
DO binary	Sets the default format to binary .

6.24 DISASSEMBLE

Abbreviation: DA

Description:

Disassembles memory contents to assembly-language code. The display of disassembled memory is fully symbolic.

Syntax:

da <address> [<length>]

Parameter	Type	Description
<address>	Numeric	Start address
<length>	Numeric	Number of instructions (optional, default = 16)

Examples:

DISASSEMBLE H’100 5	Disassembles 5 lines of code starting at H’100.
DA H’3E00 20	Disassembles 20 lines of code starting at H’3E00.

6.25 ERASE

Abbreviation: ER

Description:

Clears the **Command Line** window

Syntax:

er

Parameter	Type	Description
none		Clears the **Command Line** window

Example:

ER	Clears the **Command Line** window.

6.26 EVALUATE

Abbreviation: EV

Description:

Provides a calculator function, evaluating simple and complex expressions, with parentheses, mixed radices, and symbols. All operators have the same priority but parentheses may be used to change the order of evaluation. The operators have the same meaning as in C/C++. Expressions can also be used in any command where a number is required. Register names may be used, but must always be prefixed by the ‘#’ character. The result is displayed in hexadecimal, decimal, octal, or binary.

Syntax:

ev <expression>

Parameter	Type	Description
<expression>	Expression	Expression to be evaluated

Valid operators:

&&	logical AND	||	logical OR	<<	left arithmetic shift	>>	right arithmetic shift
+	addition	-	subtraction	*	multiplication	/	division
%	modulo	|	bitwise OR	&	bitwise AND	~	bitwise NOT
^	bitwise exclusive OR	!	logical NOT	==	equal to	!=	unequal to
>	greater than	<	less than	>=	greater than or equal to	<=	less than or equal to

Examples:

EV H'123 + (D'73 | B'10)	Result: H'16E D'366 O'556 B'00000000000000000000000101101110
EV #R1 * #R2	Result: H'121 D'289 O'441 B'00000000000000000000000100100001

6.27 FILE_LOAD

Abbreviation: FL

Description:

Loads an object code file to memory with the specified offset. Existing symbols are cleared, and the new ones are defined. If an offset is specified this will be added to the symbols. The file extension default is **.MOT**.

Syntax:

fl [<format>] <filename> [<offset>] [<state>]

Parameter	Type	Description
<format>	Keyword	Object format (optional, default = DEFAULT_OBJECT_FORMAT settings)
	Binary	Binary type
	Elf/Dwarf2	Elf/Dwarf2 type
	IntelHex	Intel-Hex type
	S-Record	S type
<filename>	Character string	File name
<offset>	Numeric	Offset to be added to load address (optional, default = 0)
<state>	Keyword	Verify flag (optional, default = V)
	V	Verify
	N	No verify

Examples:

FILE_LOAD A:\\BINARY\\TESTFILE.A22	Loads Motorola S-Record file "testfile.a22".
FL ANOTHER.MOT H’200	Loads Motorola S-Record file "another.mot" with an offset of H’200 bytes.

6.28 FILE_SAVE

Abbreviation: FS

Description:

Saves the specified memory area data to a file. The user is warned if about to overwrite an existing file. The file extension default is **.MOT**. Symbols are **not** automatically saved.

Syntax:

fs [<format>] <filename> <start> <end>

Parameter	Type	Description
<format>	Keyword	Object format (optional, default = DEFAULT_OBJECT_FORMAT settings)
	Binary	Binary type
	IntelHex	Intel-Hex type
	S-Record	S type
<filename>	Character string	File name
<start>	Numeric	Start address
<end>	Numeric	End address

Examples:

FILE_SAVE TESTFILE 0 H’2013	Saves address range 0-H’2013 as Motorola S-Record file "TESTFILE.MOT".
FS D:\\USER\\ANOTHER.A22 H’4000 H’4FFF	Saves address range H’4000-H’4FFF as Motorola S-Record format file "ANOTHER.A22".

6.29 FILE_VERIFY

Abbreviation: FV

Description:

Verifies file contents against memory contents. The file data must be in a Motorola S-Record format. The file extension default is **.MOT**.

Syntax:

fv <filename> [<offset>]

Parameter	Type	Description
<filename>	Character string	File name
<offset>	Numeric	Offset to be added to file address (optional, default = 0)

Examples:

FILE_VERIFY A:\\BINARY\\TEST.A22	Verifies Motorola S-Record file "TEST.A22" against memory.
FV ANOTHER 200	Verifies Motorola S-Record file "ANOTHER.MOT" against memory with an offset of H’200 bytes.

6.30 GO

Abbreviation: GO

Description:

Executes object code (the user program). While the user program is executing, the **Performance Analysis** window is updated. While the user system is halted, a PC value is displayed.

Syntax:

go [<state>] [<address>]

Parameter	Type	Description
<state>	Keyword	Specifies whether or not to continue command processing during user program execution (optional, default = wait)
	wait	Causes command processing to wait until user program stops
	continue	Continues command processing during execution
<address>	Numeric	Start address for PC (optional, default = PC value)

Wait is the default and this causes command processing to wait until user program stops executing.

Continue allows you to continue to enter commands (but they may not work depending on the debugging platform).

Examples:

GO	Executes the user program from the current PC value. Command processing cannot be continued.
GO CONTINUE H’1000	Executes the user program from H’1000. Command processing can be continued.

6.31 GO_RESET

Abbreviation: GR

Description:

Executes the user program starting at the address specified in the reset vector.

While the user program is executing, the **Performance Analysis** window is updated.

Syntax:

gr [<state>]

Parameter	Type	Description
<state>	Keyword	Specifies whether or not to continue command processing during user program execution (optional, default = wait)
	wait	Causes command processing to wait until user program stops
	continue	Continues command processing during execution

Wait is the default and this causes command processing to wait until user program stops executing.

Continue allows you to continue to enter commands (but they may not work depending on the debugging platform)

Example:

GR	Executes the user program starting at the address specified in the reset vector (does not continue command processing).

6.32 GO_TILL

Abbreviation: GT

Description:

Executes the user program from the current PC with temporary breakpoints. This command takes multiple addresses as parameters, and these are used to set temporary PC breakpoints (these breakpoints only exist for the duration of the command).

Syntax:

gt [<state>] <address>...

Parameter	Type	Description
<state>	Keyword	Specifies whether or not to continue command processing during user program execution (optional, default = wait)
	wait	Causes command processing to wait until user program stops
	continue	Continues command processing during execution
<address>...	Numeric	Temporary breakpoint address (list)

Wait is the default and this causes command processing to wait until user program stops executing

Continue allows you to continue to enter commands (but they may not work depending on the debugging platform)

Example:

GO_TILL H'1000	Continues execution until the PC reaches address H'1000.

6.33 HALT

Abbreviation: HA

Description:

Halts the user program. This command can be used after the GO command if the GO command uses continue for option.

Syntax:

ha

Parameter	Type	Description
none		Halts the user program

Example:

HA	Halts the user program.

6.34 INITIALIZE

Abbreviation: IN

Description:

Initializes all breakpoints and memory mapping. It also initializes debugging platform, as if you had reselected the target DLL.

Syntax:

in

Parameter	Type	Description
none		Initializes debugging platform.

Example:

IN	Initializes debugging platform.

6.35 LOG

Abbreviation: LO

Description:

Controls logging of command output to file. If no parameters are specified, logging status is displayed. If an existing file is specified, you will be warned; if you answer 'No', data will be overwritten to the existing file, otherwise the file will be added. Logging is only supported for the command line interface.

Syntax:

lo [<state>|<filename>]

Parameter	Type	Description
none		Displays logging status
<state>	Keyword	Starts or suspends logging
	+	Starts logging
	-	Suspends logging
<filename>	Numeric	Specifies the logging output file

Examples:

LOG TEST	Stores the logging in file TEST.
LO -	Suspends logging.
LOG +	Resumes logging.
LOG	Displays logging status

6.36 MAP_DISPLAY

Abbreviation: MA

Description:

Displays memory mapping.

Syntax:

ma

Parameter	Type	Description
none		Displays the current memory mapping

Example:

MA	Displays the current memory mapping.

6.37 MAP_SET

Abbreviation: MS

Description:

Allocates a memory area.

Syntax:

ms <start address> [<end address>] [<mode>]

Parameter	Type	Description
<start address>	Numeric	Specified start address
<end address>	Numeric	Specified end address (optional, default = start address)
<mode>	Keyword	Access type (optional, default = RW)
	R	Read only
	W	Write only
	RW	Displays the current memory mapping

Examples:

MAP_SET 0000 3FFF RW	A read/write-enabled area is allocated to addresses H'0000 to H'3FFF.
MS 5000	A read/write-enabled area is allocated to address H'5000.

6.38 MEMORY_DISPLAY

Abbreviation: MD

Description:

Displays memory contents.

Syntax:

md <address> [<length>] [<mode>]

Parameter	Type	Description
<address>	Numeric	Start address
<length>	Numeric	Length (optional, default = H’100 bytes)
<mode>	Keyword	Display format (optional, default = byte)
	byte	Displays in byte units
	word	Displays in word units (2 bytes)
	long	Displays in longword units (4 bytes)
	ascii	Displays in ASCII codes
	single	Displays in single-precision floating-point format
	double	Displays in double-precision floating-point format

Examples:

MEMORY_DISPLAY H’C000 H’100 WORD	Displays H’100 bytes of memory starting at H’C000 in word units
MEMORY_DISPLAY H’1000 H’FF	Displays H’FF bytes of memory starting at H’1000 in byte units

6.39 MEMORY_EDIT

Abbreviation: ME

Description:

Allows memory contents to be modified. When editing memory the current location may be modified in a similar way to that described in the **ASSEMBLE** command description.

When editing, '.' exits edit mode, '^' goes back one data unit, and blank line goes forward without modification.

Syntax:

me <address> [<mode>] [<state>]

Parameter	Type	Description
<address>	Numeric	Address to edit
<mode>	Keyword	Format (optional, default = byte)
	byte	Edits in byte units
	word	Edits in word units
	long	Edits in longword units
	ascii	Edits in ASCII codes
	single	Edits in the single-precision floating-point format
	double	Edits in the double-precision floating-point format
<state>	Keyword	Verify flag (optional, default = V)
	V	Verify
	N	No verify

Example:

ME H’1000 WORD	Modifies memory contents in word units starting from H’1000 (with verification)

6.40 MEMORY_FILL

Abbreviation: MF

Description:

Modifies the contents in the specified memory area to the specified data value.

Syntax:

mf <start> <end> <data> [<mode>] [<state>]

Parameter	Type	Description
<start>	Numeric	Start address
<end>	Numeric	End address
<data>	Numeric	Data value
<mode>	Keyword	Data size (optional, default = byte)
	byte	Byte
	word	Word
	long	Longword
	single	Single-precision floating-point
	double	Double-precision floating-point
<state>	Keyword	Verify flag (optional, default = V)
	V	Verify
	N	No verify

Examples:

MEMORY_FILL H'C000 H'C0FF H'55AA WORD	Modifies memory contents in the range from H'C000 to H'C0FF to word data H'55AA.
MF H'5000 H'7FFF H'21	Modifies memory contents in the range from H'5000 to H'7FFF to data H'21.

6.41 MEMORY_MOVE

Abbreviation: MV

Description:

Moves data in the specified memory area.

Syntax:

mv <start> <end> <dest> [<state>]

Parameter	Type	Description
<start>	Numeric	Start address
<end>	Numeric	End address (including this address)
<dest>	Numeric	Destination start address
<state>	Keyword	Verify flag (optional, default = V)
	V	Verify
	N	No verify

Examples:

MEMORY_MOVE H’1000 H’1FFF H’2000	Moves memory contents in the area from H’1000 to H’1FFF into H’2000.
MV H’FB80 H’FF7F H’3000	Moves memory contents in the area from H’FB80 to H’FF7F into H’3000.

6.42 MEMORY_TEST

Abbreviation: MT

Description:

Performs read, write, and verification testing in the specified address range. The original contents of memory have been replaced by the newly written data. The test will access the memory according to the map settings.

This simulator/debugger does not support the MEMORY_TEST command.

Syntax:

mt <start> <end>

Parameter	Type	Description
<start>	Numeric	Start address
<end>	Numeric	End address (including this address)

Examples:

MEMORY_TEST H’8000 H’BFFF	Tests from H’8000 to H’BFFF.
MT H’4000 H’5000	Tests from H’4000 to H’5000.

6.43 OPEN_WORKSPACE

Abbreviation: OW

Description:

Opens a workspace.

Syntax:

ow <filename>

Parameter	Type	Description
filename	Character string	Workspace file name

Example:

OW WKSP.HWS	Opens the WKSP.HWS file.

6.44 PROFILE

Abbreviation: PR

Description:

Enables, disables or sets the profiler display and resets the profiler information.

Syntax:

pr [<state>]

Parameter	Type	Description
None		Displays the profiler information.
<state>	Keyword	Enables, disables or sets the profiler display and resets the profiler information.
	enable	Enables the profiler.
	tree-off	Enables the profiler but does not trace function calls during profile information acquisition.
	disable	Disables the profiler.
	reset	Resets the profiler information.

Examples:

PROFILE ENABLE	Enables the profiler.
pr r	Resets the profiler information.

6.45 PROFILE_DISPLAY

Abbreviation: PD

Description:

Displays the profiler information.

Syntax:

pd [<mode>] [<state1>] [<state2>] [<count>]

Parameter	Type	Description
<mode>	Keyword	Specifies the method of displaying the profiler information. (optional, default=list)
	tree	Displays in tree format
	list	Displays in list format
<state1>	Keyword	Specifies whether or not to include child function information in the parent function cycle information. (optional, default=n)
	i	Specifies child function information to be included in the display.
	n	Specifies child function not to be included in the display.
<state2>	Keyword	Specifies whether or not to control displaying functions that are not executed (optional, default=a).
	e	Displays only executed functions.
	a	Displays all functions.
<count>	Numeric	Specifies the nesting level for calling functions to be displayed. This can be specified only when the <mode> parameter is ‘tree’ (optional, default = 16).

Examples:

PROFILE_DISPLAY TREE I	Specifies the profiler information to be displayed in tree format and to include child functions.
pd	Specifies the profiler information to be displayed in list format, without child function information.

6.46 PROFILE_SAVE

Abbreviation: PS

Description:

Saves the profiler information to a file. The default file extension is **.PRO**.

Syntax:

ps [<filename>]

Parameter	Type	Description
none		Saves the profiler information on all download modules to the files.
<filename>	Character string	Specifies the name of the file to which profiler information is saved.

Example:

PROFILE_SAVE PR_INFO	Saves profiler information to a file named PR_INFO.PRO.

6.47 QUIT

Abbreviation: QU

Description:

Exits HEW. Closes a log file if it is open.

Syntax:

qu

Parameter	Type	Description
none		Exits HEW

Example:

QU	Exits HEW.

6.48 RADIX

Abbreviation: RA

Description:

Sets default input radix. If no parameters are specified, the current radix is displayed. Radix can be changed by using B', H', D', or O' before numeric data.

Syntax:

ra [<mode>]

Parameter	Type	Description
none		Displays current radix
<mode>	Keyword	Sets radix to specified type
	H	Sets radix to hexadecimal
	D	Sets radix to decimal
	O	Sets radix to octal
	B	Sets radix to binary

Examples:

RADIX	Displays the current radix.
RA H	Sets the radix to hexadecimal.

6.49 REGISTER_DISPLAY

Abbreviation: RD

Description:

Displays CPU register contents.

Syntax:

rd

Parameter	Type	Description
none		Displays all register contents

Example:

RD	Displays all register contents

6.50 REGISTER_SET

Abbreviation: RS

Description:

Changes the contents of a register.

Syntax:

rs <register> <value> [<mode>]

Parameter	Type	Description
<register>	Keyword	Register name
<value>	Numeric	Register value
<mode>	Keyword	Data size (optional, default = corresponding register size)
	byte	Byte
	word	Word
	long	Longword
	single	Single-precision floating-point
	double	Double-precision floating-point

Examples:

RS PC_StartUp	Sets the program counter to the address defined by the symbol _StartUp
RS R0 H’1234 WORD	Sets word data H’1234 to R0.

6.51 RESET

Abbreviation: RE

Description:

Resets the microprocessor. All register values are set to the initial values of the device. Memory mapping and breakpoints are not initialized.

Syntax:

re

Parameter	Type	Description
none		Resets the microprocessor

Example:

RE	Resets the microprocessor.

6.52 RESPONSE

Abbreviation: RP

Description:

Specifies the frequency the window is updated.
When a long refresh interval is specified, the simulation becomes faster but the response, such as for the break button, becomes slower. Set a frequency appropriate for the machine used.

Syntax:

rp [<instruction number>]

Parameter	Type	Description
<instruction number>	Numeric	Specifies the frequency, in number of information executions, the window is to update. 1 to 65535 (optional, default= 40000)

Example:

RESPONSE 9	Sets the window to refresh every 9 information executions.

6.53 SLEEP

Abbreviation: None

Description:

Delays command execution for a specified period.

Syntax:

sleep <milliseconds>

Parameter	Type	Description
<milliseconds>	Numeric	Delayed time (ms)

Default radix (it is not always decimal) is used, if you do not specify D’.

Example:

SLEEP D’9000	Delays 9 seconds.

6.54 STEP

Abbreviation: ST

Description:

Single step (in source line or instruction units) execution. Performs a specified number of instructions, from current PC. Default is stepping by lines if source debugging is available. Count default is 1.

Syntax:

st [<mode>] [<count>]

Parameter	Type	Description
<mode>	Keyword	Type of single step (optional)
	instruction	Steps by assembly instruction
	line	Steps by source code line
<count>	Numeric	Number of steps (optional, default = 1)

Example:

STEP 9	Steps code for 9 steps.

6.55 STEP_MODE

Abbreviation: SM

Description:

Selects the step mode.

Syntax:

sm <mode>

Parameter	Type	Description
<mode>	Keyword	Type of step mode
	Auto	If Source windows is active, steps by source code line If Disasembly windows is active, steps by assembly instruction
	Assembly	Steps by assembly instruction
	Source	Steps by source code line

Example:

STEP_MODE atuo	Sets the step mode to auto.

6.56 STEP_OUT

Abbreviation: SP

Description:

Steps the program out of the current function. (i.e., a step up). This works for both assembly-language and source level debugging.

Syntax:

sp

Parameter	Type	Description
none		Steps the program out of the current function

Example:

SP	Steps the program out of the current function.

6.57 STEP_OVER

Abbreviation: SO

Description:

Performs a specified number of instructions from current PC.

This command differs from STEP in that it does not perform single step operation in subroutines or interrupt routines. These are executed at full speed.

Syntax:

so [<mode>] [<count>]

Parameter	Type	Description
<mode>	Keyword	Type of stepping (optional)
	instruction	Steps by assembly instruction
	line	Step by source code line
<count>	Numeric	Number of steps (optional, default = 1)

Example:

SO	Steps over 1-step code.

6.58 STEP_RATE

Abbreviation: SR

Description:

Controls the speed of stepping in the STEP and STEP_OVER commands. A rate of 6 causes the fastest stepping. A value of 0 is the slowest.

Syntax:

sr [<rate>]

Parameter	Type	Description
none		Displays the step rate
<rate>	Numeric	Step rate 0 to 6 (6 = fastest)

Examples:

SR	Displays the current step rate.
SR 6	Specifies the fastest step rate.

6.59 SUBMIT

Abbreviation: SU

Description:

Executes a file of emulator commands. This command can be used even in a command file to be processed. Any error aborts the file.

Syntax:

su <filename>

Parameter	Type	Description
<filename>	Character string	File name

Examples:

SUBMIT COMMAND.HDC	Processes the file COMMAND.HDC.
SU A:SETUP.TXT	Processes the file SETUP.TXT on drive A:.

6.60 SYMBOL_ADD

Abbreviation: SA

Description:

Adds a symbol, or changes an existing one.

Syntax:

sa <symbol> <value>

Parameter	Type	Description
<symbol>	Character string	Symbol name
<value>	Numeric	Value

Examples:

SYMBOL_ADD start H’1000	Defines the symbol start at H’1000.
SA END_OF_TABLE H’1FFF	Uses current default radix and defines END_OF_TABLE at H'1FFF.

6.61 SYMBOL_CLEAR

Abbreviation: SC

Description:

Deletes a symbol. If no parameters are specified, deletes all symbols (after confirmation).

Syntax:

sc [<symbol>]

Parameter	Type	Description
none		Deletes all symbols
<symbol>	Character string	Symbol name

Examples:

SYMBOL_CLEAR	Deletes all symbols (after confirmation).
SC start	Deletes the symbol ‘start’.

6.62 SYMBOL_LOAD

Abbreviation: SL

Description:

Loads symbols from file. File must be in XLINK Pentica-b format (i.e. 'XXXXH name'). The symbols are added to the existing symbol table.

Syntax:

sl <filename>

Parameter	Type	Description
<filename>	Character string	File name

Examples:

SYMBOL_LOAD TEST.SYM	Loads the file TEST.SYM.
SL MY_CODE.SYM	Loads the file MY_CODE.SYM.

6.63 SYMBOL_SAVE

Abbreviation: SS

Description:

Saves symbols to a file in XLINK Pentica-b format. The symbol file extension default is **.SYM**.

Syntax:

ss <filename>

Parameter	Type	Description
<filename>	Character string	File name

Examples:

SYMBOL_SAVE TEST	Saves symbol table to TEST.SYM.
SS MY_CODE.SYM	Saves the symbol table to MY_CODE.SYM.

6.64 SYMBOL_VIEW

Abbreviation: SV

Description:

Displays all defined symbols, or those containing the case sensitive string pattern.

Syntax:

sv [<pattern>]

Parameter	Type	Description
none		Displays all symbols
<pattern>	Character string	Character string that should be contained in the symbols to be displayed

Examples:

SYMBOL_VIEW BUFFER	Displays all symbols containing the word BUFFER.
SV	Displays all the symbols.

6.65 TCL

Abbreviation: None

Description:

Enables or disables the TCL.

Syntax:

tcl [<state>]

Parameter	Type	Description
None		Displays the TCL information.
<state>	Keyword	Enables or disables the TCL
	enable	Enables the TCL
	disable	Disables the TCL

Examples:

TCL	Displays the TCL information.
TCL enable	Enables the TCL.
TCL d	Disables the TCL.

6.66 TRACE

Abbreviation: TR

Description:

Displays the trace buffer contents. The record in the buffer that was executed first is 0; older records have positive offset values.

Syntax:

tr [[<start rec> [<count>]] | [<clear>]]

Parameter	Type	Description
<start rec>	Numeric	Record to start display (optional, default = most recent record - 9)
<count>	Numeric	Number of records to be displayed (optional, default = 10)
<clear>	Keyword	Clears trace records (optional)
	clear	Clears trace records

Note: When a negative value is specified for <start rec> (-0 cannot be specified), the value is recognized as a PTR value.

Examples:

TR 0 20	Displays twenty lines of trace buffer contents starting from the top of the buffer.
TR	Displays ten lines of trace buffer contents starting from the end of the buffer (the ten most recently executed lines).
TR –20 10	Displays trace buffer contents for which the PTR value is between –20 and -11.
TR C	Clears all trace records.

6.67 TRACE_ACQUISITION

Abbreviation: TA

Description:

Enables or disables trace information acquisition

Syntax:

ta <mode>

Parameter	Type	Description
<mode>	Keyword	Enabling or disabling trace information acquisition.
	E	Trace information acquisition is enabled.
	D	Trace information acquisition is disabled.

Examples:

TRACE_ACQUISITION E	Trace information acquisition is enabled.
TA D	Trace information acquisition is disabled.

6.68 TRACE_SAVE

Abbreviation: TV

Description:

Saves the trace information in files. The files are in text format, so the default file extension is **.TXT**.

Syntax:

tv<filename>

Parameter	Type	Description
<filename>	Character string	File name

Examples:

TRACE_SAVE TEST	Saves the trace information in TEST.TXT.
TV TRACE.TXT	Saves the trace information in TRACE.TXT

6.69 TRACE_STATISTIC

Abbreviation: TST

Description:

Analyzes the statistic information under the specified conditions.

Syntax:

tst <item> <string>

Parameter	Type	Description
<item>	Character string	Statistic information item to be analyzed
<string>	Character string	Character string that specifies conditions

Example:

TST CODE1 E630	Analyzes the statistic information under condition CODE1 = E630.

Section 7 Messages

7.1 Information Messages

The simulator/debugger outputs information messages as listed in table 7.1 to notify users of execution status.

Table 7.1 Information Messages

Message	Contents
Break Access	The break access condition was satisfied and execution has stopped.
Break Cycle	The break cycle condition was satisfied and execution has stopped
Break Data	The break data condition was satisfied and execution has stopped.
Break Register	The break register condition was satisfied and execution has stopped.
Break Sequence	The break sequence condition was satisfied and execution has stopped.
PC Breakpoint	The breakpoint condition was satisfied and execution has stopped.
Sleep	Execution has been stopped by the SLEEP instruction.
Step Normal End	The step execution succeeded.
Stop	Execution has been stopped by the **[Stop]** button.
Trace Buffer Full	Since the **Break** mode was selected by **Trace buffer full handling** in the **Trace Acquisition** dialog box and the trace buffer became full, execution was terminated.

7.2 Error Messages

The simulator/debugger outputs error messages to notify users of the errors of user programs or operation. Table 7.2 lists the error messages.

Table 7.2 Error Messages

Message	Contents
Address Error	One of the following states occurred: • A PC value was an odd number. • An instruction was read from the internal I/O area. • Word data was accessed to an address other than a multiple of 2. • Longword data was accessed to an address other than a multiple of 4. Correct the user program to prevent the error from occurring.
Exception Error	An error occurred during exception processing. Correct the user program to prevent the error from occurring.
File Open Error	An error occurred during opening a file with the break of file-input/output action. Correct the file setting.
File Input Error	An error occurred during reading a file with the break of file-input/output action. Correct the file setting.
File Output Error	An error occurred during writing to a file with the break of file-input/output action. Correct the file setting.
Illegal Instruction	Either of the following states occurred: • A code other than an instruction was executed. • Execution of MOV.B Rn, @-SP or MOV.B @SP + Rn was attempted. Correct the user program to prevent the error from occurring.
Illegal Operation	Either of the following states occurred: • Incorrect relation between the values in the C flag or H flag of the CCR in the DAA or DAS instruction and the values before adjustment. • Zero division executed by the DIVXU or DIVXS instruction, or overflow. Correct the user program to prevent the error from occurring.

Table 7.2 Error Messages (cont)

Message	Contents
Memory Access Error	One of the following states occurred: • A memory area that had not been allocated was accessed. • Data was written to a memory area having the write protect attribute. • Data was read from a memory area having the read disable attribute. • A memory area in which memory does not exist was accessed. • Write to the EEPROM by instructions other than EEPMOV was attempted. Allocate memory, change the memory attribute, or correct the user program to prevent the memory from being accessed.
System Call Error	System call error occurred. Modify the incorrect contents of registers R0, R1, or ER1, and parameter block.

Appendix A - GUI Command Summary

Menu	Item	Accelerator	Toolbar Graphic
View	Workspace	Alt+K	
	Output	Alt+U	
	Breakpoints	Shift+Ctrl+B	
	Coverage...	Shift+Ctrl+Q	
	Command Line	Ctrl+L	
	Disassembly	Ctrl+D	
	IO	Ctrl+I	
	Image...	Shift+Ctrl+G	
	Labels	Shift+Ctrl+A	
	Locals	Ctrl+Shift+W	
	Memory...	Ctrl+M	
	Performance Analysis	Shift+Ctrl+P	
	Profile	Shift+Ctrl+F	
	Registers	Ctrl+R	
	Status	Ctrl+U	
	Trace	Ctrl+T	
	Watch	Ctrl+W	
	Waveform...	Shift+Ctrl+V	
	Simulated I/O	Shift+Ctrl+I	
	Stack Trace	Ctrl+K	
	Trigger	Shift+Ctrl+F	

Menu	Item	Accelerator	Toolbar Graphic
Options	Debug Settings...		
	Radix ➢		
	Hex		
	Decimal		
	Oct		
	Bin		
	Simulator ➢		
	System...		
	Memory Resource...		
Debug	Reset CPU		
	Go	F5	
	Reset Go	Shift+F5	
	Go to Cursor		
	Set PC to Cursor		
	Run...		
	Step In	F11	
	Step Over	F10	
	Step Out	Shift+F11	
	Step...		
	Step Mode ➢		
	Auto		
	Assembly		
	Source		
	Halt Program	Esc	
	Initialize		
	Disconnect		
	Download Modules		
	Unload Modules		

Menu	Item	Accelerator	Toolbar Graphic
Memory	Search...		
	Copy...		
	Compare...		
	Fill...		
	Refresh		
	Configure Overlay...