

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-702-037E

Rev. 6.0

3/3/03

Hitachi, Ltd

HSS008SDIW3SE



# 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/DWARF 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 Hitachi Debugging Interface (hereinafter, referred to as the HDI).

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 HDI and 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.

The following sections, *Looking at Your Program*, *Working with Memory*, *Executing Your Program*, *Stopping Your Program*, *Looking at Variables*, *Overlay Function*, *Selecting Functions*, and *Configuring the User Interface*, provide a “how to” guide to using HDI for debugging.

This manual assumes that the HDI is used on the English version of Microsoft® Windows®95 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**

<b>CONVENTION</b>	<b>MEANING</b>
<b>[Menu-&gt;Menu Option]</b>	Bold text with '->' is used to indicate menu options (for example, <b>[File-&gt;Save As...]</b> ).
FILENAME.C	Uppercase names are used to indicate file names.
<u>"enter this string"</u>	Used to indicate text that must be entered (excluding the " " quotes).
Key+Key	Used to indicate required key presses. For example, <b>Ctrl+N</b> 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

Section 1 Overview .....	1
1.1 Features .....	1
1.2 Target User Program .....	2
1.3 Simulation Range .....	4
Section 2 System Overview.....	5
2.1 User Interface .....	5
2.2 Data Entry .....	5
2.2.1 Operators .....	5
2.2.2 Data Formats .....	5
2.2.3 Precision .....	6
2.2.4 Expression Examples .....	6
2.2.5 Symbol Format .....	6
2.2.6 Symbol Examples .....	6
2.3 Help .....	7
2.3.1 Context Sensitive Help .....	7
Section 3 Simulator/Debugger Functions.....	9
3.1 Simulator/Debugger Platforms and CPU Types.....	9
3.2 Simulator/Debugger Memory Management.....	9
3.2.1 Memory Map Specification.....	9
3.2.2 Memory Resource Specification .....	10
3.3 Instruction Execution Reset Processing .....	10
3.4 Exception Processing .....	11
3.5 Features Specific to H8S/2600 CPU .....	11
3.6 Control Registers.....	12
3.7 Trace.....	12
3.8 Standard I/O and File I/O Processing .....	13
3.9 Calculation of Instruction Execution Cycles .....	25
3.10 Break Conditions.....	25
3.10.1 Break Due to the Satisfaction of a Break Command Condition.....	26
3.10.2 Break Due to the Detection of an Error During Execution of the User Program .....	26
3.10.3 Break Due to a Trace Buffer Overflow .....	28
3.10.4 Break Due to Execution of the SLEEP Instruction .....	28
3.10.5 Break Due to the [STOP] Button .....	29
3.11 Floating-Point Data .....	29
3.12 Display of Function Call History .....	29
Section 4 Menus .....	31

4.1	File .....	31
4.1.1	New Session.....	31
4.1.2	Load Session.....	31
4.1.3	Save Session.....	32
4.1.4	Save Session As... ..	32
4.1.5	Load Program.....	32
4.1.6	Initialize.....	32
4.1.7	Exit .....	32
4.2	Edit .....	32
4.2.1	Cut.....	32
4.2.2	Copy.....	33
4.2.3	Paste .....	33
4.2.4	Find... ..	33
4.2.5	Evaluate.....	33
4.3	View .....	33
4.3.1	Breakpoints .....	33
4.3.2	Command Line.....	34
4.3.3	Disassembly... ..	34
4.3.4	Labels .....	34
4.3.5	Locals .....	34
4.3.6	Memory.....	34
4.3.7	Performance Analysis .....	34
4.3.8	Profile-List .....	34
4.3.9	Profile-Tree .....	35
4.3.10	Registers.....	35
4.3.11	Source.....	35
4.3.12	Status .....	35
4.3.13	Trace.....	35
4.3.14	Watch .....	35
4.3.15	Simulated I/O .....	35
4.3.16	Stack Trace.....	35
4.3.17	External Tool.....	36
4.4	Run .....	36
4.4.1	Reset CPU .....	36
4.4.2	Go.....	36
4.4.3	Reset Go.....	36
4.4.4	Go To Cursor.....	36
4.4.5	Set PC To Cursor.....	36
4.4.6	Run.....	36
4.4.7	Step In .....	36
4.4.8	Step Over.....	37
4.4.9	Step Out.....	37
4.4.10	Step.....	37

4.4.11	Halt.....	37
4.5	Memory.....	37
4.5.1	Refresh .....	37
4.5.2	Load... ..	37
4.5.3	Save.....	38
4.5.4	Verify... ..	38
4.5.5	Test.....	38
4.5.6	Fill... ..	38
4.5.7	Copy... ..	38
4.5.8	Compare... ..	38
4.5.9	Configure Map... ..	39
4.5.10	Configure Overlay.....	39
4.6	Setup.....	39
4.6.1	Status Bar .....	39
4.6.2	Options.....	39
4.6.3	Radix .....	39
4.6.4	Customize.....	39
4.6.5	Configure Platform.....	40
4.7	Window .....	40
4.7.1	Cascade .....	40
4.7.2	Tile .....	40
4.7.3	Arrange Icons .....	40
4.7.4	Close All.....	40
4.8	Help .....	41
4.8.1	Index.....	41
4.8.2	Using Help .....	41
4.8.3	Search for Help on .....	41
4.8.4	About HDI.....	41
<b>Section 5 Windows and Dialog Boxes .....</b>		<b>43</b>
5.1	Breakpoints Window.....	43
5.1.1	Add.....	44
5.1.2	Edit.....	44
5.1.3	Delete .....	44
5.1.4	Delete All .....	44
5.1.5	Disable/Enable .....	44
5.1.6	Go To Source .....	44
5.2	Set Break Dialog Box.....	45
5.3	Break Sequence Dialog Box.....	46
5.4	Command Line Window .....	47
5.4.1	Set Batch File... ..	48
5.4.2	Play.....	48
5.4.3	Set Log File.....	48

5.4.4	Logging .....	48
5.4.5	Select All.....	48
5.4.6	Copy.....	48
5.5	Disassembly Window.....	48
5.5.1	Copy.....	50
5.5.2	Set Address.....	50
5.5.3	Go To Cursor.....	50
5.5.4	Set PC Here .....	50
5.5.5	Instant Watch... ..	50
5.5.6	Add Watch .....	50
5.5.7	Go to Source.....	50
5.6	Labels Window .....	51
5.6.1	Add.....	52
5.6.2	Edit.....	52
5.6.3	Find.....	53
5.6.4	Find Next.....	53
5.6.5	View Source.....	53
5.6.6	Copy.....	53
5.6.7	Delete .....	53
5.6.8	Delete All .....	54
5.6.9	Load.....	54
5.6.10	Save.....	55
5.6.11	Save As.....	55
5.7	Locals Window .....	56
5.7.1	Copy.....	56
5.7.2	Edit Value.....	56
5.7.3	Radix .....	57
5.8	Memory Window .....	57
5.8.1	Refresh .....	57
5.8.2	Load.....	57
5.8.3	Save.....	58
5.8.4	Test.....	58
5.8.5	Fill.....	58
5.8.6	Copy.....	58
5.8.7	Compare.....	58
5.8.8	Search.....	59
5.8.9	Set Address.....	59
5.8.10	ASCII/Byte/Word/Long/Single Float/Double Float.....	59
5.9	Performance Analysis Window .....	59
5.9.1	Add Range.....	60
5.9.2	Edit Range.....	60
5.9.3	Delete Range .....	60
5.9.4	Reset Counts/Times.....	60

5.9.5	Delete All Ranges.....	60
5.9.6	Enable Analysis.....	60
5.10	Performance Option Dialog Box.....	61
5.11	Registers Window .....	62
5.11.1	Copy .....	62
5.11.2	Edit... .....	62
5.11.3	Toggle Bit.....	63
5.12	Source Window .....	63
5.12.1	Copy .....	64
5.12.2	Find... .....	64
5.12.3	Set Address.....	64
5.12.4	Set Line... .....	64
5.12.5	Go To Cursor.....	64
5.12.6	Set PC Here .....	65
5.12.7	Instant Watch... .....	65
5.12.8	Add Watch .....	65
5.12.9	Go To Disassembly .....	65
5.13	System Status Window.....	65
5.13.1	Update .....	66
5.13.2	Copy .....	66
5.14	Trace Window .....	66
5.14.1	Find... .....	67
5.14.2	Find Next.....	67
5.14.3	Filter... .....	67
5.14.4	Acquisition.....	67
5.14.5	Halt.....	68
5.14.6	Restart .....	68
5.14.7	Snapshot.....	68
5.14.8	Clear .....	68
5.14.9	Save.....	68
5.14.10	View Source .....	68
5.14.11	Trim Source.....	68
5.15	Trace Acquisition Dialog Box.....	69
5.16	Trace Search Dialog Box .....	70
5.17	Watch Window.....	71
5.17.1	Copy .....	71
5.17.2	Delete .....	71
5.17.3	Delete All .....	72
5.17.4	Add Watch... .....	72
5.17.5	Edit Value.....	72
5.17.6	Radix .....	72
5.18	System Configuration Dialog Box .....	72
5.19	Memory Map Modify Dialog Box .....	74

5.20	Memory Map Dialog Box .....	75
5.21	System Memory Resource Modify Dialog Box .....	76
5.22	Control Registers Window .....	77
5.23	SYSCR Dialog Box .....	78
5.24	Simulated I/O Window .....	79
5.25	Stack Trace Window .....	80
	5.25.1 Copy .....	80
	5.25.2 Go to Source .....	80
	5.25.3 View Setting .....	81
5.26	Profile-List Window .....	82
	5.26.1 View Source .....	82
	5.26.2 View Profile-Tree .....	82
	5.26.3 View Profile-Chart .....	83
	5.26.4 Enable Profiler .....	83
	5.26.5 Find .....	83
	5.26.6 Clear Data .....	83
	5.26.7 Output Profile Information File .....	83
	5.26.8 Output Text File .....	84
	5.26.9 Select Data .....	84
	5.26.10 Setting .....	84
5.27	Profile-Tree Window .....	85
	5.27.1 View Source .....	86
	5.27.2 View Profile-List .....	86
	5.27.3 View Profile-Chart .....	86
	5.27.4 Enable Profiler .....	86
	5.27.5 Find .....	87
	5.27.6 Find Data .....	87
	5.27.7 Clear Data .....	87
	5.27.8 Profile Information File .....	87
	5.27.9 Output Text File .....	87
	5.27.10 Select Data .....	88
	5.27.11 Setting .....	88
5.28	Profile-Chart Window .....	89
	5.28.1 Expands Size .....	89
	5.28.2 Reduces Size .....	89
	5.28.3 View Source .....	90
	5.28.4 View Profile-List .....	90
	5.28.5 View Profile-Tree .....	90
	5.28.6 View Profile-Chart .....	90
	5.28.7 Enable Profiler .....	90
	5.28.8 Clear Data .....	91
	5.28.9 Multiple View .....	91
	5.28.10 Output Profile Information File .....	91

Section 6 Command Lines.....	93
!(COMMENT) .....	95
ANALYSIS .....	95
ANALYSIS_RANGE .....	96
ANALYSIS_RANGE_DELETE .....	96
ASSEMBLE.....	97
ASSERT .....	97
BREAKPOINT .....	98
BREAK_ACCESS .....	98
BREAK_CLEAR .....	99
BREAK_DATA .....	100
BREAK_DISPLAY .....	100
BREAK_ENABLE .....	101
BREAK_REGISTER .....	102
BREAK_SEQUENCE .....	103
DISASSEMBLE .....	103
ERASE.....	104
EVALUATE .....	104
FILE_LOAD .....	105
FILE_SAVE.....	106
FILE_VERIFY .....	106
GO .....	107
GO_RESET.....	108
GO_TILL .....	109
HALT .....	109
HELP .....	110
INITIALISE.....	110
LOG .....	111
MAP_DISPLAY .....	112
MAP_SET.....	112
MEMORY_DISPLAY .....	113
MEMORY_EDIT.....	114
MEMORY_FILL .....	115
MEMORY_MOVE.....	116
MEMORY_TEST .....	116
QUIT .....	117
RADIX.....	117
REGISTER_DISPLAY .....	118
REGISTER_SET .....	119
RESET .....	120
SLEEP.....	120
STEP .....	121
STEP_OUT .....	121

STEP_OVER .....	122
STEP_RATE.....	122
SUBMIT .....	123
SYMBOL_ADD .....	123
SYMBOL_CLEAR.....	124
SYMBOL_LOAD.....	124
SYMBOL_SAVE.....	125
SYMBOL_VIEW.....	125
TRACE .....	126
TRACE_ACQUISITION.....	126

**Section 7 Messages ..... 129**

7.1 Information Messages .....	129
7.2 Error Messages.....	130

**Section 8 Looking at Your Program ..... 131**

8.1 Compiling for Debugging .....	131
8.2 Viewing the Code.....	131
8.2.1 Viewing Source Code.....	131
8.2.2 Viewing Assembly-Language Code.....	132
8.2.3 Modifying Assembly-Language Code .....	132
8.3 Looking at Labels.....	133
8.3.1 Listing Labels.....	133
8.3.2 Adding a Label from a Source or Disassembly Window .....	134
8.4 Looking at a Specific Address.....	135
8.4.1 Looking at the Current Program Counter Address.....	135
8.5 Finding Text.....	136

**Section 9 Working with Memory..... 137**

9.1 Looking at an Area of Memory .....	137
9.1.1 Displaying Memory as ASCII.....	138
9.1.2 Displaying Memory as Bytes .....	138
9.1.3 Displaying Memory as Words.....	138
9.1.4 Displaying Memory as Longwords .....	138
9.1.5 Displaying Memory as Single-Precision Floating Point .....	138
9.1.6 Displaying Memory as Double-Precision Floating Point.....	138
9.1.7 Looking at a Different Area of Memory .....	138
9.2 Modifying Memory Contents.....	139
9.2.1 Quick Edit .....	139
9.2.2 Full Edit.....	139
9.2.3 Selecting a Memory Range .....	140
9.3 Finding a Value in Memory .....	140
9.4 Filling an Area of Memory with a Value .....	141



9.4.1	Filling a Range .....	141
9.5	Copying an Area of Memory.....	142
9.6	Saving an Area of Memory .....	142
9.7	Loading an Area of Memory.....	143
9.8	Verifying an Area of Memory.....	143
<b>Section 10 Executing Your Program .....</b>		<b>145</b>
10.1	Running from Reset.....	145
10.2	Continuously Running Your Program.....	145
10.3	Running to the Cursor .....	146
10.4	Running to Several Points .....	146
10.5	Single Step.....	146
10.5.1	Stepping Into a Function .....	147
10.5.2	Stepping Over a Function Call .....	147
10.6	Stepping Out of a Function .....	147
10.7	Multiple Steps .....	147
<b>Section 11 Stopping Your Program.....</b>		<b>149</b>
11.1	Halting Execution.....	149
11.2	Standard Breakpoints (PC Breakpoints).....	149
11.3	The Breakpoints Window.....	150
11.3.1	Adding a Breakpoint .....	151
11.3.2	Modifying a Breakpoint .....	151
11.3.3	Deleting a Breakpoint.....	151
11.3.4	Deleting All Breakpoints.....	152
11.4	Disabling Breakpoints .....	152
11.4.1	Disabling a Breakpoint.....	152
11.4.2	Enabling a Breakpoint.....	152
11.5	Temporary Breakpoints .....	152
<b>Section 12 Looking at Variables .....</b>		<b>155</b>
12.1	Tooltip Watch.....	155
12.2	Instant Watch.....	155
12.3	Using Watch Items .....	156
12.3.1	Adding a Watch.....	156
12.3.2	Expanding a Watch .....	157
12.3.3	Modifying Radix for Watch Item Display.....	158
12.3.4	Changing a Watch Item's Value .....	158
12.3.5	Deleting a Watch.....	159
12.4	Looking at Local Variables .....	159
12.5	Looking at Registers.....	160
12.5.1	Expanding a Bit Register.....	160
12.5.2	Modifying Register Contents .....	161

12.5.3 Using Register Contents .....	162
<b>Section 13 Overlay Function.....</b>	<b>163</b>
13.1 Displaying Section Group .....	163
13.2 Setting Section Group .....	164
<b>Section 14 Selecting Functions .....</b>	<b>165</b>
14.1 Displaying Functions.....	165
14.2 Specifying Functions.....	166
14.2.1 Selecting a Function .....	166
14.2.2 Deleting a Function .....	166
14.2.3 Setting a Function .....	166
<b>Section 15 Configuring the User Interface.....</b>	<b>167</b>
15.1 Arranging Windows .....	167
15.1.1 Minimizing Windows.....	167
15.1.2 Arranging Icons.....	168
15.1.3 Tiling Windows.....	169
15.1.4 Cascading Windows.....	169
15.2 Locating Currently Open Windows.....	169
15.2.1 Locating the Next Window .....	170
15.2.2 Locating a Specific Window .....	170
15.3 Enabling/Disabling the Status Bar.....	170
15.4 Customizing the Toolbar .....	171
15.4.1 Overall Appearance.....	171
15.4.2 Customizing Individual Toolbars .....	172
15.4.3 Button Categories.....	173
15.4.4 Adding a Button to a Toolbar.....	173
15.4.5 Positioning a Button in a Toolbar.....	173
15.4.6 Removing a Button from a Toolbar .....	173
15.5 Customizing the Fonts.....	174
15.6 Customizing the File Filters .....	174
15.7 Saving a Session.....	176
15.8 Loading a Session.....	176
15.9 Setting HDI Options .....	177
15.10 Setting the Default Input Radix .....	178
<b>Appendix A - System Modules .....</b>	<b>181</b>
<b>Appendix B - GUI Command Summary.....</b>	<b>183</b>
<b>Appendix C - Symbol File Format.....</b>	<b>187</b>

# Figures

Figure 1.1	Creation of Target User Programs .....	3
Figure 4.1	Menus.....	31
Figure 5.1	Breakpoints Window.....	43
Figure 5.2	Set Break Dialog Box.....	45
Figure 5.3	Break Sequence Dialog Box .....	46
Figure 5.4	Command Line Window .....	47
Figure 5.5	Disassembly Window.....	49
Figure 5.6	Labels Window .....	51
Figure 5.7	Add Label Dialog Box .....	52
Figure 5.8	Edit Label Dialog Box .....	52
Figure 5.9	Find Label Containing Dialog Box .....	53
Figure 5.10	Message Box for Confirming Label Deletion .....	54
Figure 5.11	Message Box for Confirming All Label Deletion .....	54
Figure 5.12	Load Symbols Dialog Box .....	55
Figure 5.13	Locals Window .....	56
Figure 5.14	Memory Window .....	57
Figure 5.15	Performance Analysis Window.....	59
Figure 5.16	Performance Option Dialog Box.....	61
Figure 5.17	Registers Window .....	62
Figure 5.18	Source View.....	63
Figure 5.19	System Status Window .....	65
Figure 5.20	Trace Window.....	66
Figure 5.21	Trace Acquisition Dialog Box.....	69
Figure 5.22	Trace Search Dialog Box .....	70
Figure 5.23	Watch Window .....	71
Figure 5.24	System Configuration Dialog Box .....	72
Figure 5.25	Memory Map Modify Dialog Box .....	74
Figure 5.26	Memory Map Dialog Box .....	75
Figure 5.27	System Memory Resource Modify Dialog Box .....	76
Figure 5.28	Control Registers Window .....	77
Figure 5.29	SYSCR Dialog Box.....	78
Figure 5.30	Simulated I/O Window .....	79
Figure 5.31	Stack Trace Window .....	80
Figure 5.32	Stack Trace Setting Dialog Box .....	81
Figure 5.33	Profile-List Window.....	82
Figure 5.34	Warning Message Box Showing Profiler and Analysis Cannot Be Set at a Time .....	83
Figure 5.35	Setting Profile-List Dialog Box.....	84
Figure 5.36	Profile-Tree Window .....	85
Figure 5.37	Warning Message Box Showing Profiler and Analysis Cannot Be Set at a Time .....	86
Figure 5.38	Find Data Dialog Box .....	87
Figure 5.39	Setting Profile-Tree Dialog Box .....	88

Figure 5.40	Profile-Chart Window .....	89
Figure 5.41	Warning Message Box Showing Profiler and Analysis Cannot Be Set at a Time .....	90
Figure 8.1	Source Window .....	131
Figure 8.2	Disassembly Window.....	132
Figure 8.3	Assembler Dialog Box .....	133
Figure 8.4	Labels Window .....	134
Figure 8.5	Label Dialog Box .....	134
Figure 8.6	Set Address Dialog Box .....	135
Figure 8.7	Find Dialog Box.....	136
Figure 9.1	Open Memory Window Dialog Box .....	137
Figure 9.2	Memory Window (Bytes) .....	137
Figure 9.3	Set Address Dialog Box .....	139
Figure 9.4	Edit Dialog Box .....	140
Figure 9.5	Search Memory Dialog Box.....	140
Figure 9.6	Fill Memory Dialog Box .....	141
Figure 9.7	Copy Memory Dialog Box.....	142
Figure 9.8	Save Memory As Dialog Box .....	142
Figure 9.9	Load Memory Dialog Box .....	143
Figure 9.10	Verify S-Record File with Memory Dialog Box .....	144
Figure 10.1	Highlighted Line Corresponding to PC Address.....	145
Figure 10.2	Step Program Dialog Box .....	148
Figure 11.1	Setting a Program Breakpoint .....	150
Figure 11.2	Breakpoints Window.....	151
Figure 11.3	Run Program Dialog Box.....	153
Figure 12.1	Tooltip Watch .....	155
Figure 12.2	Instant Watch Dialog Box .....	156
Figure 12.3	Add Watch Dialog Box.....	157
Figure 12.4	Watch Window .....	157
Figure 12.5	Expanding a Watch .....	158
Figure 12.6	Edit Value Dialog Box .....	159
Figure 12.7	Locals Window .....	159
Figure 12.8	Registers Window .....	160
Figure 12.9	Expanding a Bit Register.....	161
Figure 12.10	Register Dialog Box .....	162
Figure 13.1	Overlay Dialog Box (at Opening) .....	163
Figure 13.2	Overlay Dialog Box (Address Range Selected) .....	163
Figure 13.3	Overlay Dialog Box (Highest-Priority Section Group Selected) .....	164
Figure 14.1	Select Function Dialog Box .....	165
Figure 15.1	Minimizing a Window .....	167
Figure 15.2	Disassembly Window Icon .....	168
Figure 15.3	Icons Before Arrangement .....	168
Figure 15.4	Icons After Arrangement.....	169
Figure 15.5	Selecting a Window .....	170

Figure 15.6	Customize Toolbar (Toolbars) Dialog Box.....	171
Figure 15.7	Customize Toolbar (Commands) Dialog Box.....	172
Figure 15.8	Font Dialog Box.....	174
Figure 15.9	Customize File Filter Dialog Box.....	175
Figure 15.10	Session Name Display.....	176
Figure 15.11	HDI Options (Session) Dialog Box.....	177
Figure 15.12	HDI Options (Confirmation) Dialog Box.....	178
Figure 15.13	HDI Options (Viewing) Dialog Box.....	178
Figure 15.14	Setting Radix.....	179
Figure A.1	HDI System Modules.....	181

## Tables

Table 3.1	Simulator/Debugger Platforms and CPU Types.....	9
Table 3.2	Memory Types.....	9
Table 3.3	I/O Functions.....	13
Table 3.4	Processing When a Break Condition is Satisfied.....	26
Table 3.5	Simulation Errors.....	27
Table 3.6	Register States at Simulation Error Stop.....	28
Table 6.1	Simulator/Debugger Commands.....	93
Table 7.1	Information Messages.....	129
Table 7.2	Error Messages.....	130



# Section 1 Overview

The Hitachi Debugging Interface (HDI) 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

- **Windows® GUI for debugging**
- **Intuitive interface**
- **On-line help**
- **Common “Look & Feel”**

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

The simulator/debugger provides simulation functions for H8S and H8/300 series microcomputers (H8/300, H8/300L, H8/300H, H8S/2600, and H8S/2000 series) 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 and H8/300 series C/C++ compiler
- H8S and 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 performs a simulation to calculate the number of instruction execution cycles for a program, 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 and H8/300 series CPUs
  - Functions to trace instructions or subroutines

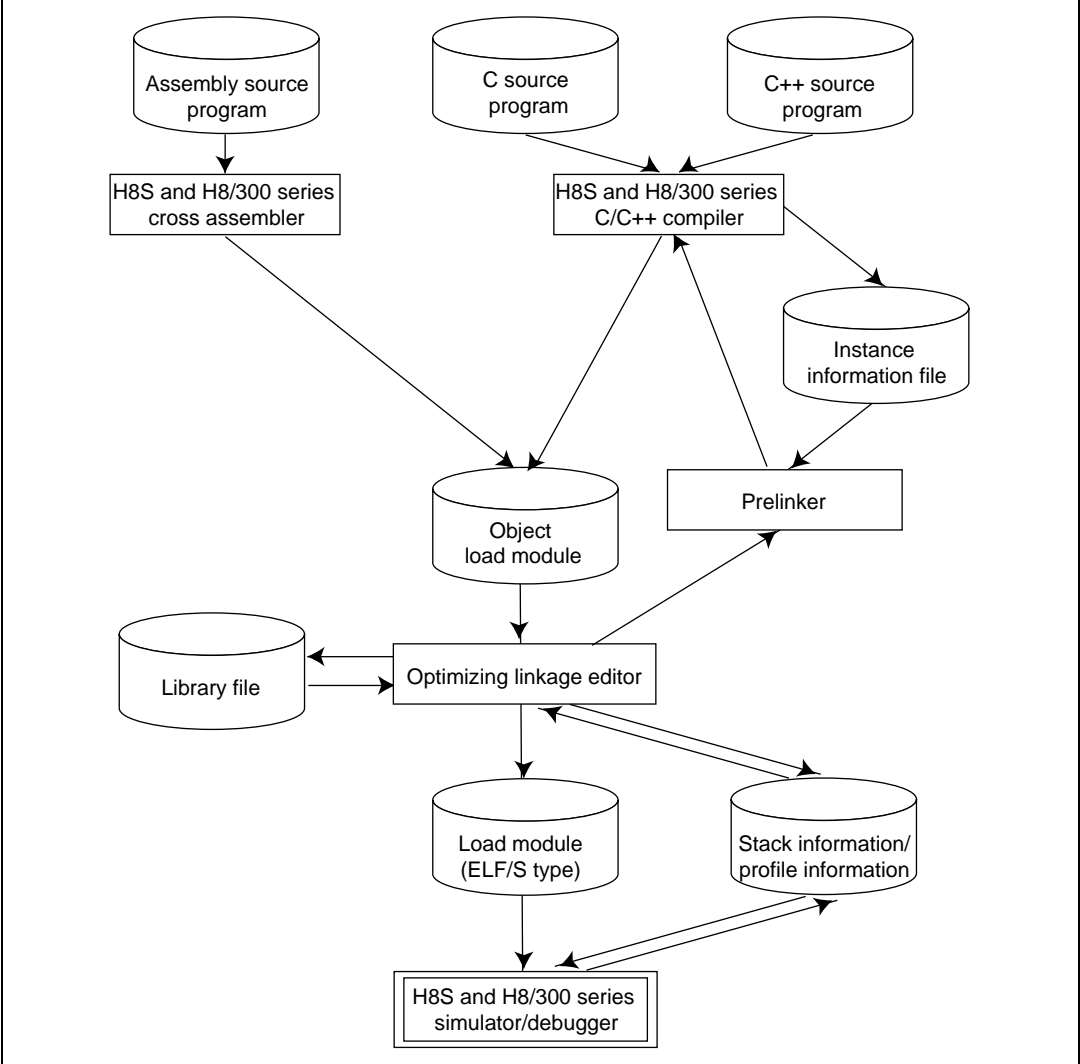
- Functions to stop or continue execution when an error occurs during user program execution
- Profile data acquisition and function-unit performance measurement
- A comprehensive set of break functions
- Functions to set or edit memory maps
- Functions to display function call history
- The breakpoint, memory map, performance, and trace can be set through the dialog box under Windows<sup>®</sup>. Environments corresponding to each memory map of the H8S and H8/300 microcomputers can be set through the dialog box.

## **1.2 Target User Program**

Load modules in ELF/DWARF 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

The simulator/debugger supports the following H8S and H8/300 series microcomputer functions:

- All CPU instructions
- Exception processing
- Registers
- All address areas

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

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

# Section 2 System Overview

HDI 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 HDI Graphical User Interface is a Windows<sup>®</sup> application that presents the debugging platform to you and allows you to set up and modify the system. Refer to a standard Windows<sup>®</sup> user manual for details on how to operate within a Windows<sup>®</sup> 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 an object DLL 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 [**Setup->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:

O'    Octal  
B'    Binary

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, #ER3, #R4L
```

### 2.2.3 Precision

All mathematics in expression evaluation is done using 32 bits (signed). Any values exceeding 32 bits are truncated.

### 2.2.4 Expression Examples

```
Buffer_start + 0x1000
#R1 | B'10001101
((pointer + (2 * increment_size)) & H'FFFF0000) >> D'15
!(flag ^ #ER4)
```

### 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++)

## 2.3 Help

HDI has a standard Windows® context sensitive help system. This provides on-line information about using the debugging system.

Help can be invoked by pressing the **F1** key or via the Help menu. Additionally, some windows and dialog boxes have a dedicated help button to launch the help file at the appropriate content.

### 2.3.1 Context Sensitive Help

To get help on a specific item in the HDI help cursor can be used. To enable the help cursor, press **SHIFT+F1** or click the button on tool bar.

Your cursor then changes to include a question mark. You can then click on the item for which you require help and the help system will be opened at the appropriate content.



# Section 3 Simulator/Debugger Functions

This section describes the H8S and H8/300 series simulator/debugger functions.

## 3.1 Simulator/Debugger Platforms and CPU Types

Each simulator/debugger platform corresponds to a specific type of CPU as listed in table 3.1.

Select a correct platform from the **Select Session** dialog box.

**Table 3.1 Simulator/Debugger Platforms and CPU Types**

Simulator/Debugger Platform	CPU Type
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

## 3.2 Simulator/Debugger Memory Management

### 3.2.1 Memory Map Specification

A memory map is specified to calculate the number of memory access cycles during simulation. The simulator/debugger supports the memory types shown in table 3.2.

**Table 3.2 Memory Types**

Memory Type	User Program Execution
Internal ROM	Enabled
Internal RAM	Enabled
External memory	Enabled
Internal I/O	Disabled
EEPROM	Enabled

A memory map can be specified in the **System Configuration** 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 5.18, System Configuration Dialog Box. The user program can be executed in all areas except for the internal I/O area.

### 3.2.2 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 **System Memory Resource Modify** 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.

However, unlike the other memory areas, a read-only EEPROM can be written to with the EEPMOV instruction and a writable EEPROM can be written to only with the EEPMOV instruction.

## 3.3 Instruction Execution Reset Processing

The simulator/debugger resets the instruction execution count and instruction execution cycles 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.



## 3.4 Exception Processing

The simulator/debugger detects the exceptions corresponding to TRAPA instructions (only for the H8/300H and H8S series) and traces (only for the H8S series) and then 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 register in the stack area. If the validity bit in EXR is set to 1, the simulator/debugger also saves the EXR register. If an error occurs while saving registers, the simulator/debugger stops exception processing, displays that an exception processing error has occurred, and enters the command input wait state.
3. Sets the I bit in the CCR register as 1.
4. Reads the start address from the vector address corresponding to the vector number. If an error occurs while reading the address, the 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.

## 3.5 Features Specific to H8S/2600 CPU

**MAC Instruction:** The H8S/2600 CPU performs multiply and accumulate operation (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:

MACS bit = 0: Non-saturation operation

MACS bit = 1: Saturation operation

**EXR Register:** The H8S/2600 CPU has the EXR register, which can be enabled or disabled by the state of bit 5 (the EXR bit) in the SYSCR register in the internal I/O:

EXR bit = 0: EXR disabled

EXR bit = 1: EXR enabled

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

**Note:** The SYSCR register address must be within the internal I/O; 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 5.18, System Configuration Dialog Box, section 5.22, Control Registers Window, and section 5.23, SYSCR Dialog Box.

### 3.6 Control Registers

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

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

To modify or display the control register value, use the **Control Registers** window and the **SYSCR** dialog box. For details, refer to section 5.18, System Configuration Dialog Box, section 5.22, Control Registers Window, and section 5.23, SYSCR Dialog Box.

### 3.7 Trace

The simulator/debugger writes the results of each instruction execution into the trace buffer. The trace buffer can hold the results for up to 1024 instruction executions. 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 is as follows.

- Total number of instruction execution cycles
- Instruction address
- CCR
- Multiplier internal flag (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 5.14, Trace Window through section 5.16, Trace Search Dialog Box.

## 3.8 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 3.3 to be executed by the user program. When the I/O processing is executed, the **Simulated I/O** window must be open.

Table 3.3 lists the I/O processing functions supported by the simulator/debugger. Each function has three types of function codes: those for the 16-bit address, for the 24-bit address, and for the 32-bit address. Select one according to the target CPU.

**Table 3.3 I/O Functions**

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

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

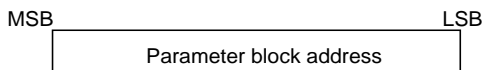
1. Set the address specialized for I/O processing in the [System Call Address], select [Enable] and execute the program.
2. When detecting a subroutine call instruction (BSR or JSR), that is, a system call to the specialized address during user program execution, the simulator/debugger performs I/O processing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

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

- Set the function code (table 3.3) to the R0 register



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

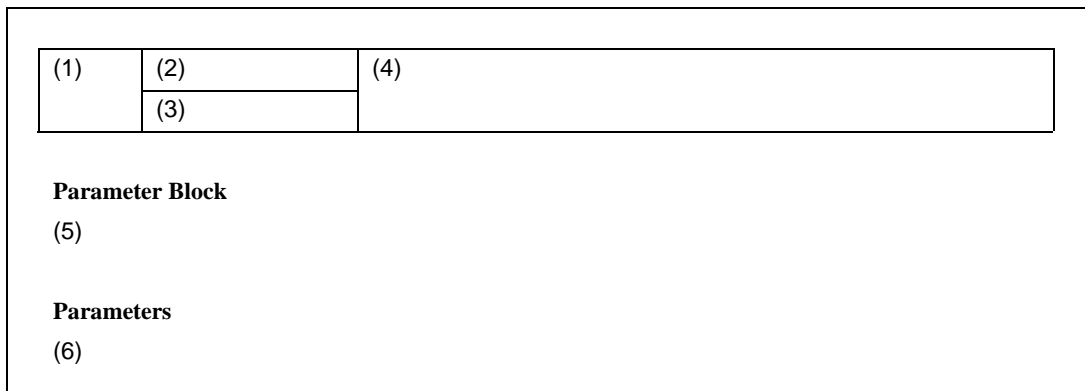


- Reserve the parameter block and input/output buffer areas

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:

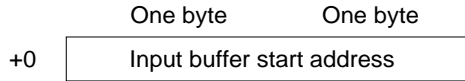


- (1) Number corresponding to table 3.3
- (2) Function name
- (3) Function code
- (4) I/O overview
- (5) I/O parameter block
- (6) I/O parameters

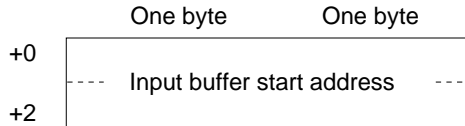
1	GETC	Inputs one byte from the standard input device
	H'01, H'11, H'21	

### Parameter Block

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



- Function code: H'11 (for the 24-bit address) or H'21 (for the 32-bit address)



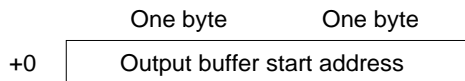
### Parameters

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

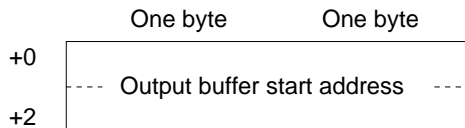
2	PUTC	Outputs one byte to the standard output device
	H'02, H'12, H'22	

### Parameter Block

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



- Function code: H'12 (for the 24-bit address) or H'22 (for the 32-bit address)



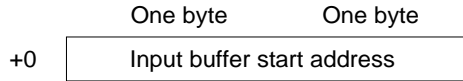
### Parameters

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

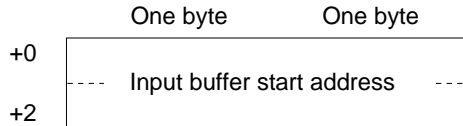
3	GETS	Inputs one line from the standard input device
	H'03, H'13, H'23	

### Parameter Block

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



- Function code: H'13 (for the 24-bit address) or H'23 (for the 32-bit address)



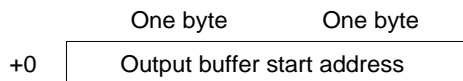
### Parameters

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

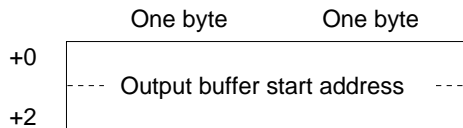
4	PUTS	Outputs one line to the standard output device
	H'04, H'14, H'24	

### Parameter Block

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



- Function code: H'14 (for the 24-bit address) or H'24 (for the 32-bit address)



### 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, 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 (for the 16-bit address)

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

- Function code: H'15 (for the 24-bit address) or H'25 (for the 32-bit address)

	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.

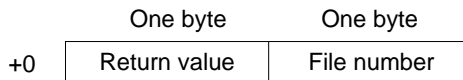
"b" : Open in binary mode.

- 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)

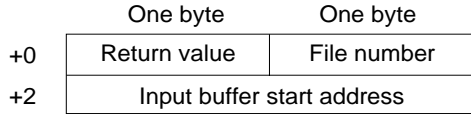
The number returned when the file was opened.



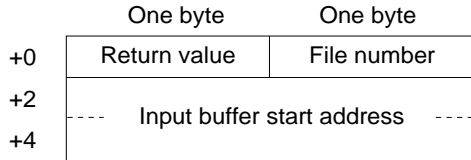
7	FGETC	Inputs one byte from a file
	H'07, H'17, H'27	

### Parameter Block

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



- Function code: H'17 (for the 24-bit address) or H'27 (for the 32-bit address)



### 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, H'28	

### Parameter Block

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

	One byte	One byte
+0	Return value	File number
+2	Output buffer start address	

- Function code: H'18 (for the 24-bit address) or H'28 (for the 32-bit address)

	One byte	One byte
+0	Return value	File number
+2	----- Output buffer start address -----	
+4		

### 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, 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 (for the 16-bit address)

	One byte	One byte
+0	Return value	File number
+2	Buffer size	
+4	Input buffer start address	

- Function code: H'19 (for the 24-bit address) or H'29 (for the 32-bit address)

	One byte	One byte
+0	Return value	File number
+2	Buffer size	
+4	Input buffer start address	
+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, 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 (for the 16-bit address)

	One byte	One byte
+0	Return value	File number
+2	Output buffer start address	

- Function code: H'1A (for the 24-bit address) or H'2A (for the 32-bit address)

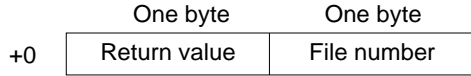
	One byte	One byte
+0	Return value	File number
+2	----- Output buffer start address -----	
+4		

### 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

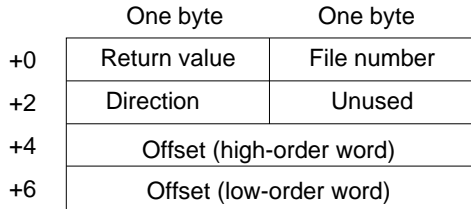


### Parameters

- 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



### 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 (high-order word)	
+4	Offset (low-order 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). As the system call address, label SYS\_CALL is specified:

```

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

```

## 3.9 Calculation of Instruction Execution Cycles

The simulator/debugger calculates the number of instruction execution cycles using the expression described in the H8S series or H8/300 series programming manual, and the data bus width and the number of access cycles specified in the **System Configuration** dialog box. However, the calculated 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 count the number of execution cycles of standard I/O and file I/O processing because the processing is specific to the simulator/debugger. Here, standard I/O and file I/O processing begins when execution has jumped, due to the BSR or JSR instruction, to an address specified in **[System Call Address]** and ends when execution has returned to the calling address.

## 3.10 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

### 3.10.1 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

When a break condition is satisfied during user program execution, the instruction at the breakpoint may or may not be executed before a break depending on the type of break, as listed in table 3.4.

**Table 3.4 Processing When a Break Condition is Satisfied**

<b>Command</b>	<b>Instruction When a Break Condition is Satisfied</b>
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.

### 3.10.2 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 **System Configuration** dialog box specifies whether to stop or continue the simulation when such an error occurs. Table 3.5 lists the error messages, error causes, and the action of the simulator/debugger in the continuation mode.



**Table 3.5 Simulation Errors**

<b>Error Message</b>	<b>Error Cause</b>	<b>Processing in Continuation Mode</b>
Address Error	A PC value was an odd number.	Operates in the same way as the actual device.
	An instruction was fetched from the internal I/O area.	
	Word data was accessed to an odd-numbered address.	
	Longword data was accessed to an odd-numbered address.	
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	
	Data was written to EEPROM with an instruction other than EEPMOV.	
Illegal Instruction	A code other than an instruction was executed.	Always stops.
	MOV.B Rn, @-SP or MOV.B @SP+, Rn was executed.	Continues operation; the result is not guaranteed.
Illegal Operation	In the DAA or DAS instruction, relationship between the C and H flags of CCR, and their relation to the value before compensation were incorrect.	Continues operation; the result is not guaranteed.
	Zero-division or overflow was caused by the DIVXU or DIVXS instruction.	

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 3.6 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 3.6 Register States at Simulation Error Stop**

Error Message	PC Value
Address Error, Memory Access Error	<ul style="list-style-type: none"> <li>When an instruction is read: The start address of the instruction that caused the error.</li> <li>When an instruction is executed: The instruction address following the instruction that caused the error.</li> </ul>
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.

- First execute the program in the stop mode and confirm that there are no errors except those in the intended locations.
- 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.**

### 3.10.3 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 when execution is stopped.

```
Trace Buffer Full
```

### 3.10.4 Break Due to Execution of the SLEEP Instruction

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

```
Sleep
```

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

### 3.10.5 Break Due to the [STOP] Button

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

Stop

Execution can be resumed with the GO or STEP command.

## 3.11 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 **System Configuration** 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.

## 3.12 Display of Function Call History

The simulator/debugger displays the function call history in the **Stack Trace** window when simulation stops, which enables 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.

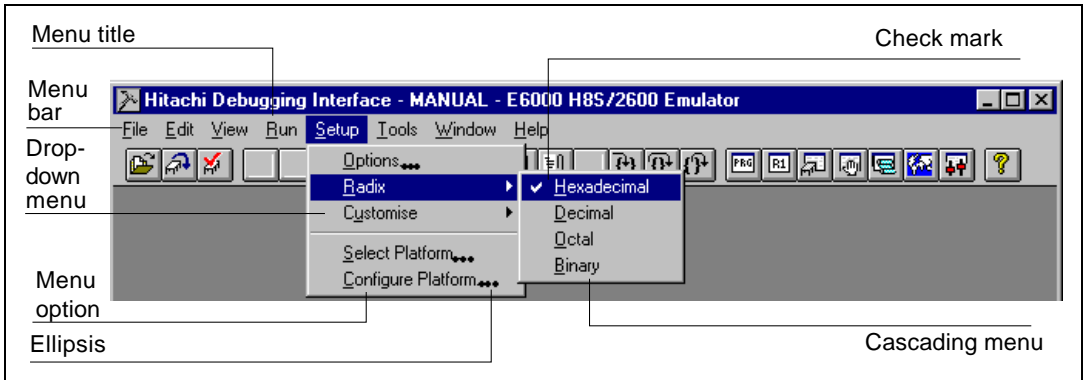
The displayed function call history is updated in the following cases:

- When simulation stops under the break conditions described in section 3.10, 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 5.25, Stack Trace Window.

# 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.

## 4.1 File

The File menu is used for aspects of the program that access program files.

### 4.1.1 New Session...



Launches the **Select Session** dialog box allowing the user to select a new debugging platform.

### 4.1.2 Load Session...



Launches the **Select Session** dialog box allowing the user to load a session from a selected session file (\*.hds extension). A session file contains the debugging platform's settings, and the current program and the position of open child windows (views) - it contains symbols, breakpoints, or current register values.

### 4.1.3 Save Session



Updates the session file for the current session file. If there is no current session file defined, this acts in a similar manner to the [**Save Session As...**] menu option.

### 4.1.4 Save Session As...

Launches the **Save As** dialog box allowing the user to save the current session details under a new file name. A session file contains the debugging platform's settings, and the current program and the position of open child windows (views) - it contains symbols, breakpoints, or current register values.

### 4.1.5 Load Program...



Launches the **Load Program** dialog box, allowing the user to select an object file in either S-Record (\*.mot; \*.s20; and \*.obj extensions) or ELF/DWARF (\*.abs extension) format and download it to the debugging platform's memory. This will also load the symbols if they are available in the selected file.

### 4.1.6 Initialize



This will attempt to re-initialize the debugging system. It will close down any open child windows and shut down the link to the debugging platform. If this is successful, an attempt to re-establish the link to the debugging platform will be made. The message 'Link up' will appear in the left-most box of the status bar if this is successful. (See also section 4.4.1, Reset CPU)

### 4.1.7 Exit

This will close down the HDI. The actions that are carried out by the HDI can be defined by the user in the 'On Exit' section of the **HDI Options** dialog box. (See also section 4.6.2, Options...)

## 4.2 Edit

The Edit menu is used for aspects of the program that access or modify data in the child windows and debugging platform.

### 4.2.1 Cut



Only available if a block is highlighted in a child window whose contents can be modified.

This will remove the contents of the highlighted block from the window and place it on the clipboard in the standard Windows® manner.

#### 4.2.2 Copy



Only available if a block is highlighted in a child window whose contents can be modified. This will copy the contents of the highlighted block to the clipboard in the standard Windows® manner.

#### 4.2.3 Paste



Only available if the contents of the child window can be modified. This will copy the contents of the Windows® clipboard into the child window at the current cursor position.

#### 4.2.4 Find...



Only available if the window contains text. This will launch the **Find** dialog box allowing the user to enter a word and locate occurrences within the text. If a match is found, the cursor will move to the start of the word.

#### 4.2.5 Evaluate...



Launches the **Evaluate** dialog box allowing the user to enter a numeric expression, e.g. " $(\#PC + 205) * 2$ ", and display the result in all currently supported radices.

### 4.3 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.

#### 4.3.1 Breakpoints



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

### 4.3.2 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.

### 4.3.3 Disassembly...



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

### 4.3.4 Labels



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

### 4.3.5 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.

### 4.3.6 Memory...



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

### 4.3.7 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.

### 4.3.8 Profile-List



Opens the **Profile-List** 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.



### 4.3.9 Profile-Tree



Opens the **Profile-Tree** window allowing the user to view the relation of function calls in a tree structure. The **Profile-Tree** window also displays the address, size, and stack size of each function, number of function calls, and profile data. The stack size, number of function calls, and profile data are values when the function is called.

### 4.3.10 Registers



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

### 4.3.11 Source...



Launches the **Open** dialog box allowing the user to enter a file name of the source file (in either C/C++ or assembly language format) to view. If the source file is not included within the current program or there is no debugging information for the file within the 'absolute' (\*.abs) file, then the message "Cannot load program. No Source level debugging available" is displayed.

### 4.3.12 Status



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

### 4.3.13 Trace



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

### 4.3.14 Watch



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

### 4.3.15 Simulated I/O

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

### 4.3.16 Stack Trace

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

### 4.3.17 External Tool

Opens the **External Tools** window allowing the user to use the co-verification tool.

## 4.4 Run

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

### 4.4.1 Reset CPU



Resets the user system hardware and sets the PC to the reset vector address. (See also section 4.1.6, Initialize).

### 4.4.2 Go



Starts executing the user program at the current PC.

### 4.4.3 Reset Go



Executes the user program from the reset vector address.

### 4.4.4 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.

### 4.4.5 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.

### 4.4.6 Run...

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

### 4.4.7 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 4.4.10, Step...). If a subroutine call is reached, then the subroutine will be entered and the view is updated to include its code.

#### 4.4.8 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 4.4.10, 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.

#### 4.4.9 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.

#### 4.4.10 Step...



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

#### 4.4.11 Halt



Stops the execution of the user program.

### 4.5 Memory

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

#### 4.5.1 Refresh

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

#### 4.5.2 Load...



Launches the **Load Memory** dialog box, allowing the user to select an offset address in the memory area, and file name to load from an S-Record format file on disk.

#### 4.5.3 Save...



Launches the **Save Memory As** dialog box, allowing the user to select a start and an end address in the memory area, to save to an S-Record format file on disk. 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.

#### 4.5.4 Verify...



Launches the **Verify S-Record File with Memory** dialog box, allowing the user to select a start and an end address in the memory area to check against the contents of an S-Record file on disk.

#### 4.5.5 Test...



Launches the **Test Memory** dialog box allowing the user to specify a block of memory to test for correct read/write operation. The exact test is target dependent. However, in all cases the current contents of the memory will be overwritten - **YOUR PROGRAM AND DATA WILL BE ERASED**. This simulator/debugger does not support this function.

#### 4.5.6 Fill...



Launches the **Fill Memory** dialog box allowing the user to fill a block of the debugging platform's memory with a value. The start and end fields can be specified in the same way as that with the Save option (refer to section 4.5.3, Save...).

#### 4.5.7 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. The start and end fields can be specified in the same way as that with the Save option (refer to section 4.5.3, Save...).

#### 4.5.8 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. The start and end fields can be specified in the same way as that with the Save option (refer to section 4.5.3, Save...).

## 4.5.9 Configure Map...



Opens the **Memory Mapping** window allowing the user to view and edit the debugging platform's current memory map. In some debugging platforms, the **Memory Map** dialog box will open.

## 4.5.10 Configure Overlay...



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

# 4.6 Setup

The Setup menu is used to modify the settings of the HDI user interface, and the configuration of the debugging platform.

## 4.6.1 Status Bar

Toggles the status bar feature on and off. If the feature is enabled then a check mark will be displayed to the left of the menu text.

## 4.6.2 Options...



Launches the **HDI Options** dialog box allowing the user to modify the settings that are specific to the HDI (not debugging platform dependent settings).

## 4.6.3 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 check mark to its left and the associated toolbar button 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.

## 4.6.4 Customize



Cascades a menu displaying a list of options that can be customized by the user.

**Toolbar** :When this cascade menu option is selected, the **Customize** dialog box is launched.

**Font** :When this cascade menu option is selected, the **Font** dialog box is launched, allowing a fixed width font to be selected.

**File Filter** : When this cascade menu option is selected, the **Customize File Filter** dialog box is launched, allowing the browser file filters for object, source and memory files to be changed to match the user's requirements.

#### 4.6.5 Configure Platform...



Launches a **set-up** dialog box allowing the user to modify the debugging platform settings. Refer to section 5.18, System Configuration Dialog Box for more details.

### 4.7 Window

The Window menu modifies the display of currently open child windows. The following menu options are always displayed, and a numbered list of current child windows will be appended - the topmost child window will have a check mark.

#### 4.7.1 Cascade



Arranges the child windows in the standard cascade manner, i.e. from the top left such that the title bar of each child window is visible.

#### 4.7.2 Tile



Arranges the child windows in the standard tile manner, i.e. sizes each window such that all are displayed without overlapping.

#### 4.7.3 Arrange Icons



Lines up any iconized windows neatly along the bottom of the parent frame in the standard manner.

#### 4.7.4 Close All

Closes all the child windows.

## 4.8 Help

The Help menu accesses additional information on how to use the functionality provided by HDI.

### 4.8.1 Index



Opens the main help file at the index.

### 4.8.2 Using Help

Opens a help file allowing the user to find out how to use Windows® hypertext help system.

### 4.8.3 Search for Help on

Opens the main help file and launches the **Search** dialog box allowing the user to enter and browse through the file's keywords.

### 4.8.4 About HDI

Launches the **About HDI** dialog box allowing the user to view the version of HDI and the currently loaded DLLs.

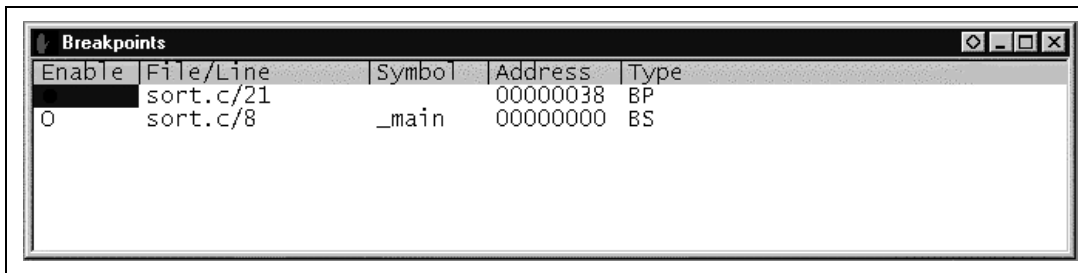




# Section 5 Windows and Dialog Boxes

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 Breakpoints Window



**Figure 5.1 Breakpoints 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. Breakpoints with mark ● or ○ are enabled.

[File/Line] Displays file names and line numbers where breakpoints are specified.

[Symbol] Displays symbols that correspond to breakpoint setting addresses. When no symbol exists, nothing is displayed.

[Address] Displays addresses where breakpoints are specified.

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

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

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

### 5.1.1 Add...

Sets a breakpoint. Launches the **Set Break** dialog box allowing the user to set a break condition.

### 5.1.2 Edit...

Only enabled if a breakpoint is selected. Launches the **Set Break** dialog box allowing the user to modify the properties of an existing breakpoint. When [Break Sequence] is selected, the **Break Sequence** dialog box opens.

### 5.1.3 Delete

Only enabled if a breakpoint is selected. 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 5.1.5, **Disable/Enable**).

### 5.1.4 Delete All

Removes all breakpoints from the list.

### 5.1.5 Disable/Enable

Only enabled if a breakpoint is selected. Toggles the selected breakpoint between enabled and disabled (when disabled, a breakpoint remains in the list, but does not cause a break when the specified conditions are satisfied). When a breakpoint is enabled, a check mark is shown to the left of the menu text (and a circle is shown in the **Enable** column for the breakpoint).

### 5.1.6 Go To Source

Opens **Source** or **Disassembly** window at address of breakpoint.

## 5.2 Set Break Dialog Box

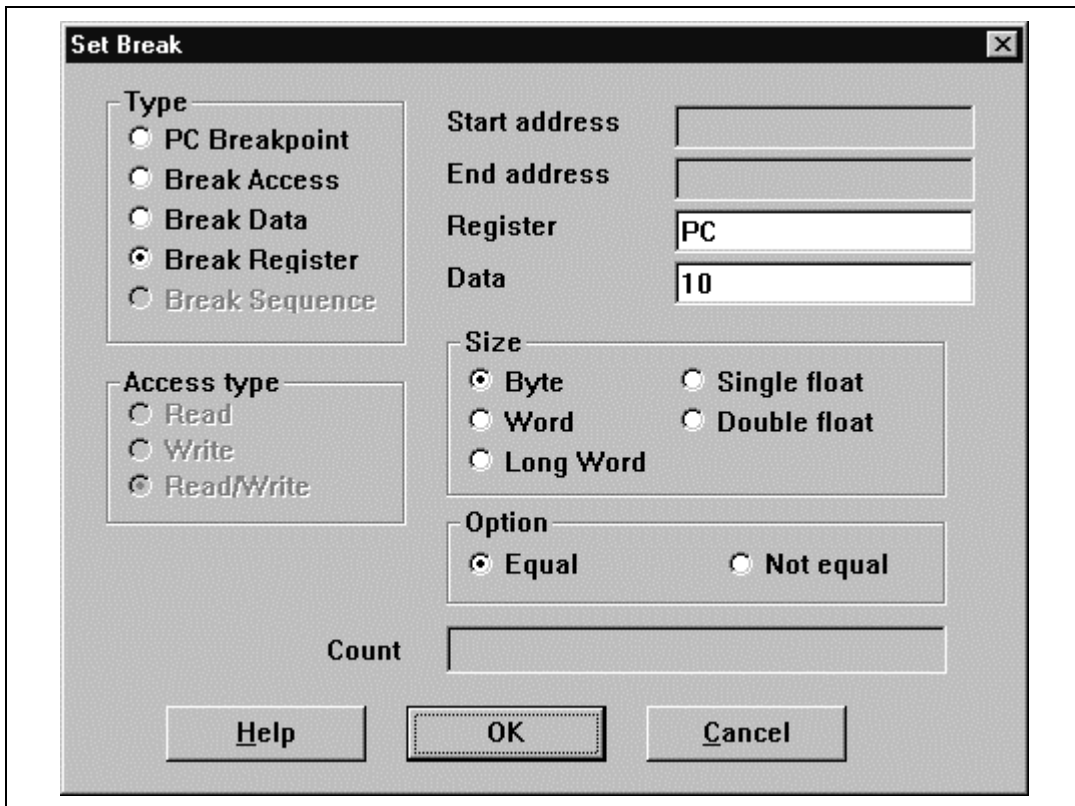


Figure 5.2 Set Break Dialog Box

This dialog box specifies break conditions.

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

[PC Breakpoint]	[Start address]	Address where a break occurs
	[Count]	Number of times that a specified instruction is fetched (default: 1)
[Break Access]	[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

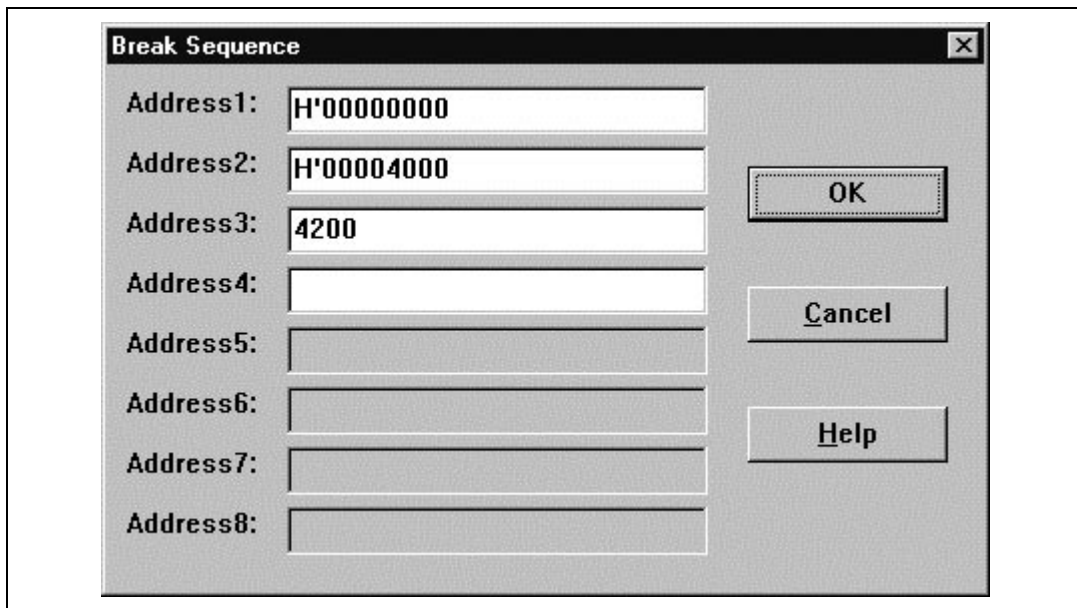
[Break Data]	[Start Address]	Address of memory where a break occurs
	[Data]	Data value that causes a break
	[Size]	Data size
	[Option]	Data match/mismatch
[Break Register]	[Register]	Register name where break conditions are specified
	[Data]	Data value that causes a break (If no data is input, a break occurs whenever data is written to the register)
	[Size]	Data size
	[Option]	Data match/mismatch

Note that when **[Break Sequence]** is selected under **[Type]**, the **Break Sequence** dialog box opens.

When **[PC Breakpoint]** is selected, if an overloaded function or class name including a member function is specified in **[Start Address]**, the **Select Function** dialog box opens. In the dialog box, select a function. For details, refer to section 14, Selecting Functions.

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

### 5.3 Break Sequence Dialog Box



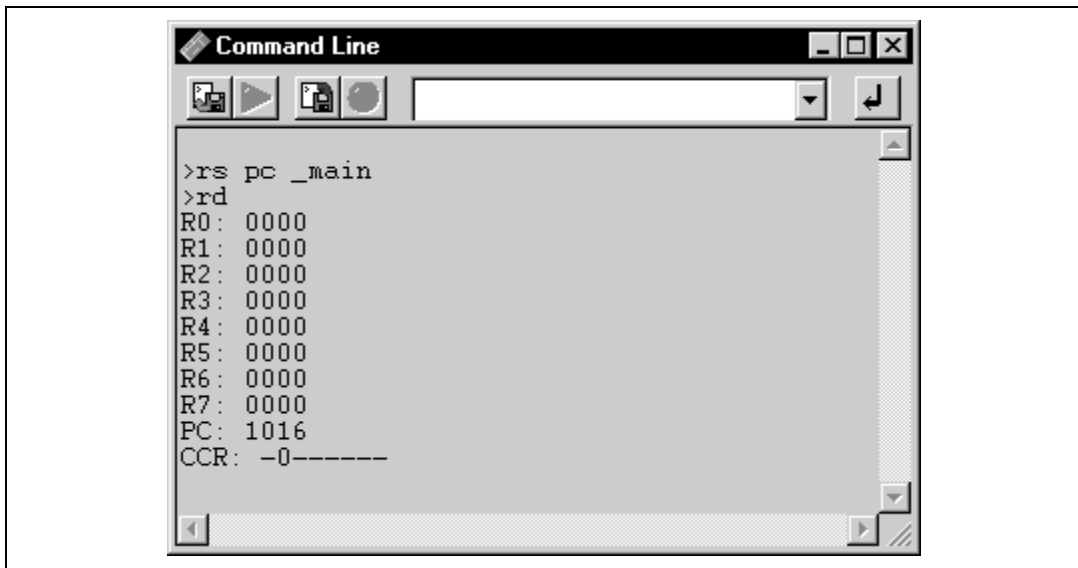
**Figure 5.3 Break Sequence Dialog Box**

This dialog box specifies the pass addresses as break conditions.

Specify addresses in [Address1] to [Address8]. Not all eight addresses need to be specified. When an overloaded function or a class name including a member function is specified as a pass address, the **Select Function** dialog box will open; select the function name in the dialog box. For details, refer to section 14, Selecting Functions.

Clicking the [OK] button sets the pass addresses. Clicking the [Cancel] button closes this dialog box without adding a new pass address.

## 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 text box (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.

The functionality of the toolbar buttons is identical to the popup menu options shown below.

### 5.4.1 Set Batch File...



Launches the **Set Batch File** dialog box, allowing the user to enter the name of an HDI command file (\*.hdc). The batch file is then run automatically. The name of the file is shown on the window title bar.

### 5.4.2 Play



Runs the last executed HDI command file (\*.hdc). It is displayed in gray while the batch file is running and the display is enabled when the command file execution is terminated and the control is returned to the user.

### 5.4.3 Set Log File...



Launches the **Open Log File** dialog box, allowing the user to enter the name of an HDI log file (\*.log). The logging option is automatically set and the name of the file is 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.

### 5.4.4 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.

### 5.4.5 Select All

Selects all the output from the **Command Line** window.

### 5.4.6 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.

## 5.5 Disassembly Window

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

This window layout has a different layout from the **Source** window, with an additional column Label which displays the symbol/label name (if available) for that address. 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.

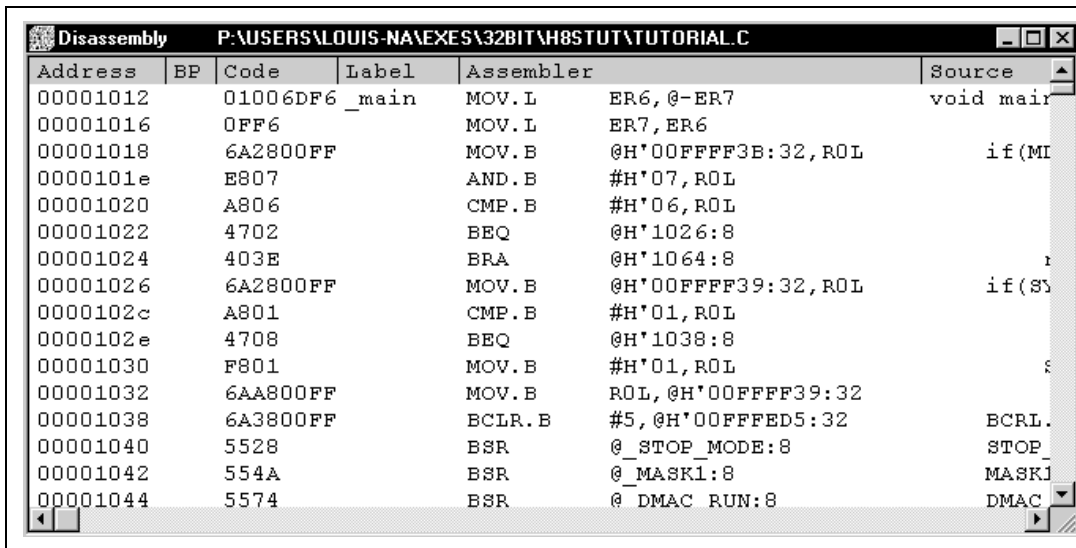


Figure 5.5 Disassembly Window

Each column supports column-specific double-click actions:

- BP - Toggles standard breakpoint at that address.
- Address - Launches the **Set Address** dialog box, allowing the user to enter a new address. If the address is in a source file, then that file will be opened in a new window (a current source view will be brought into focus) with the cursor set to the specified address. Finally, if the address does not correspond to a source file, then this window will scroll to that location. When an overloaded function or a class name is entered in the Set Address edit field, the **Select Function** dialog box opens for you to select a function.
- Code and Assembler - 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 any changes will be lost at the end of the session.
- Label - Launches the **Label** dialog box, allowing the user to enter a new label, or to clear or edit the name of an existing label.
- Source - Launches editor at location in source (set by optional startup parameters in Windows® Start menu HDI shortcut).

Within the BP column a list of currently supported standard breakpoint types can be displayed by right clicking. The currently selected standard breakpoint is shown by a check mark to the left of the menu text.

A popup menu containing the following options is available by right clicking within the window, but outside the BP column:

### 5.5.1 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.

### 5.5.2 Set Address...

Launches the **Set Address** dialog box, allowing the user to enter a new start address. 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.

### 5.5.3 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.

### 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 Instant Watch...

Launches the **Instant Watch** dialog box with the name extracted from the view at the current text cursor (not mouse cursor) position. Only valid in the selected source column.

### 5.5.6 Add Watch

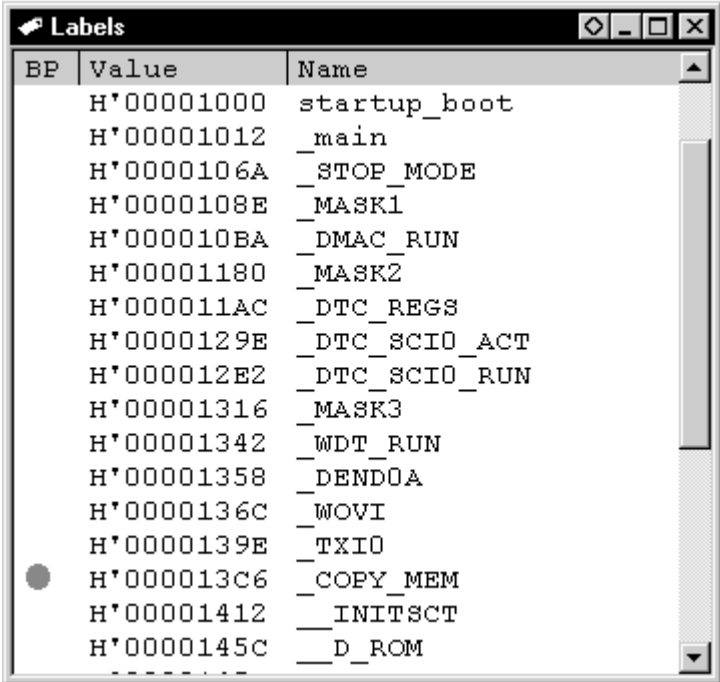
Adds the name extracted from the view at the current text cursor (not mouse cursor) position to the list of watched variables. If a **Watch** window is not open, then it is opened and brought to the top of the child windows. Only valid in the source column.

### 5.5.7 Go to Source

Opens the **Source** window corresponding to the current text cursor (not mouse cursor) position. Only valid in the source column.



## 5.6 Labels Window



The screenshot shows a window titled "Labels" with a table containing memory addresses, values, and names. The table has three columns: BP, Value, and Name. A checkmark is visible in the BP column for the entry with Value H'000013C6.

BP	Value	Name
	H'00001000	startup_boot
	H'00001012	_main
	H'0000106A	_STOP_MODE
	H'0000108E	_MASK1
	H'000010BA	_DMAC_RUN
	H'00001180	_MASK2
	H'000011AC	_DTC_REGS
	H'0000129E	_DTC_SCIO_ACT
	H'000012E2	_DTC_SCIO_RUN
	H'00001316	_MASK3
	H'00001342	_WDT_RUN
	H'00001358	_DEND0A
<input checked="" type="checkbox"/>	H'0000136C	_WOVI
<input type="checkbox"/>	H'0000139E	_TXIO
<input type="checkbox"/>	H'000013C6	_COPY_MEM
<input type="checkbox"/>	H'00001412	_INITSCT
<input type="checkbox"/>	H'0000145C	_D_ROM

**Figure 5.6 Labels 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:

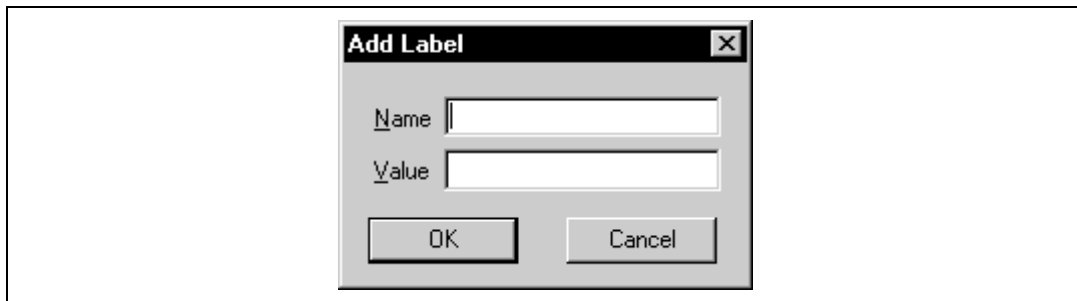
- BP – Sets or cancels a standard breakpoint at that address.
- Address - Opens a **Source** window at the start of the function.
- Name - Launches the **Edit Label** dialog box.

Within the BP column a list of currently supported standard breakpoint types can be displayed by right clicking. The currently selected standard breakpoint is shown by a check mark to the left of the menu text.

A popup menu containing the following options is available by right clicking within the window, but outside the BP column:

## 5.6.1 Add...

Launches the **Add Label** dialog box:

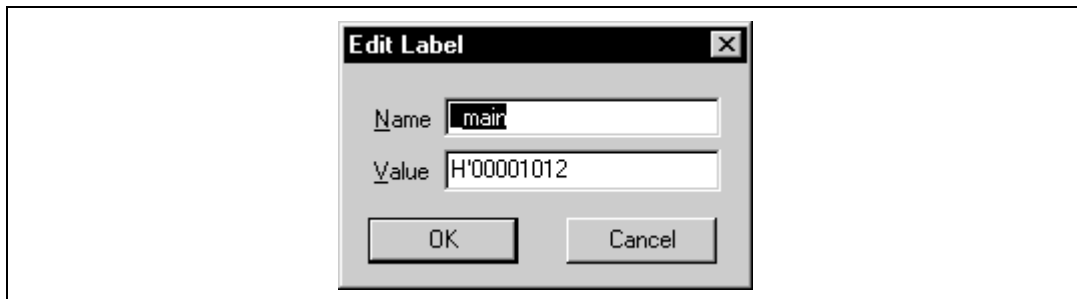


**Figure 5.7 Add Label Dialog Box**

Enter the new label name into the Name field and the corresponding value into the Value 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 Name field, the **Select Function** dialog box opens for you to select a function. For details, refer to section 14, Selecting Functions.

## 5.6.2 Edit...

Launches the **Edit Label** dialog box:

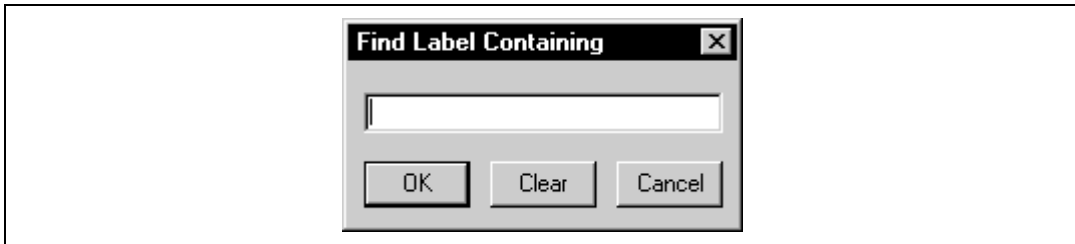


**Figure 5.8 Edit Label Dialog Box**

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 Name field, the **Select Function** dialog box opens for you to select a function. For details, refer to section 14, Selecting Functions.

### 5.6.3 Find...

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



**Figure 5.9 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 HDI 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.

### 5.6.4 Find Next

After a label has been searched, searches for the next label that matches the search condition.

### 5.6.5 View Source

Opens the **Source** or the **Disassembly** window that matches the address of the label.

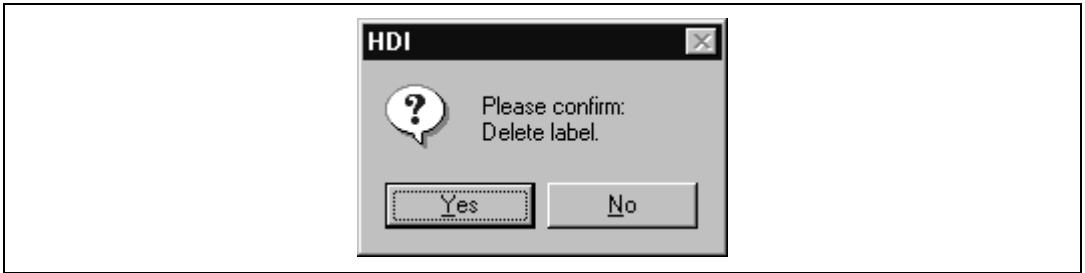
### 5.6.6 Copy



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.

### 5.6.7 Delete

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



**Figure 5.10 Message Box for Confirming Label Deletion**

If you click on the [Yes] 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 tab in the **HDI Options** dialog box.

### 5.6.8 Delete All

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

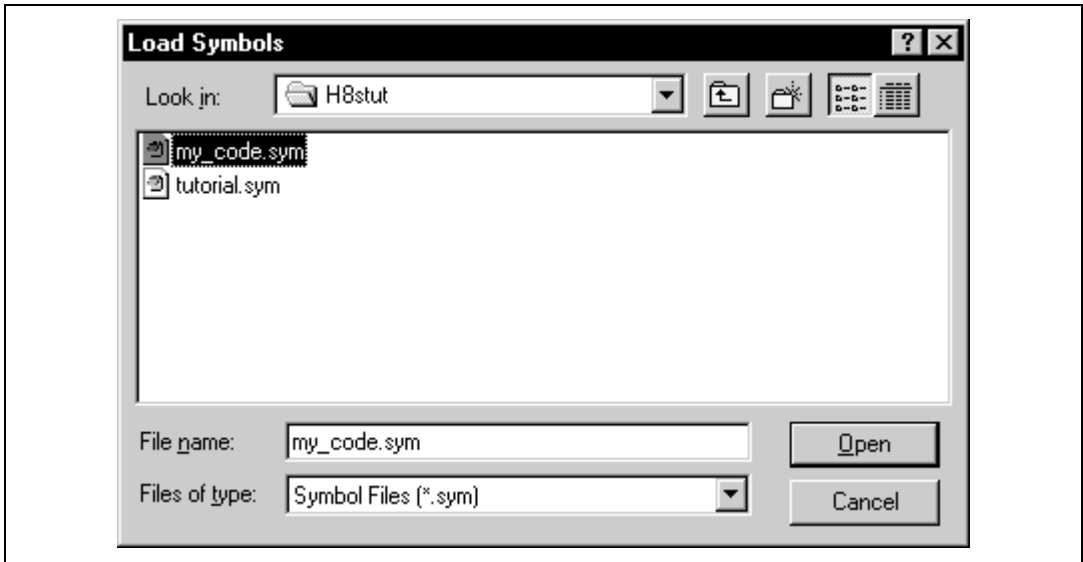


**Figure 5.11 Message Box for Confirming All Label Deletion**

If you click on the [Yes] button all the labels are removed from the HDI 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 tab in the **HDI Options** dialog box.

### 5.6.9 Load...

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



**Figure 5.12 Load Symbols Dialog Box**

The dialog box operates like a standard Windows<sup>®</sup> 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 will be displayed showing how many symbols have been loaded (this can be switched off in the Confirmations tab on the HDI Options dialog).

### 5.6.10 Save

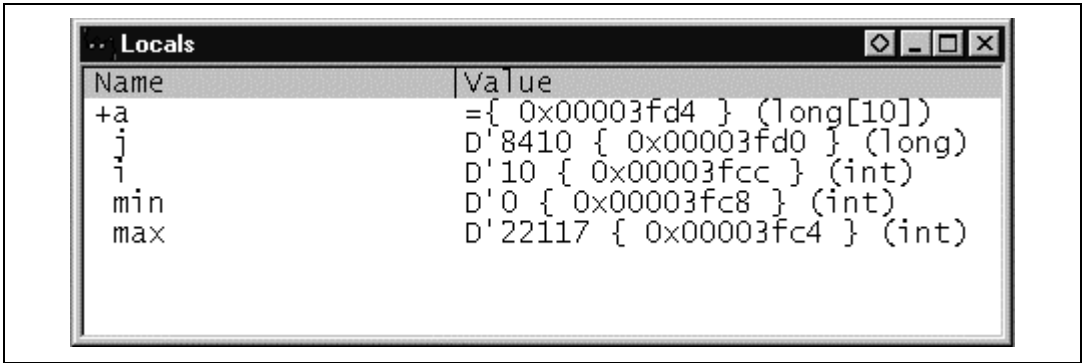
Saves HDI's current symbol table to a symbol file.

### 5.6.11 Save As...

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

See appendix C for symbol file format.

## 5.7 Locals Window



**Figure 5.13 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 object file.

The following items are displayed:

[Name] Variable name

[Value] Value, allocated position, and variable type. The allocated position is enclosed in { }, and the type is enclosed in ( ).

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 reduced. Alternatively, the plus and minus keys may be used. For more information on the display of information, refer to section 12.3.2, Expanding a Watch.

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

### 5.7.1 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.

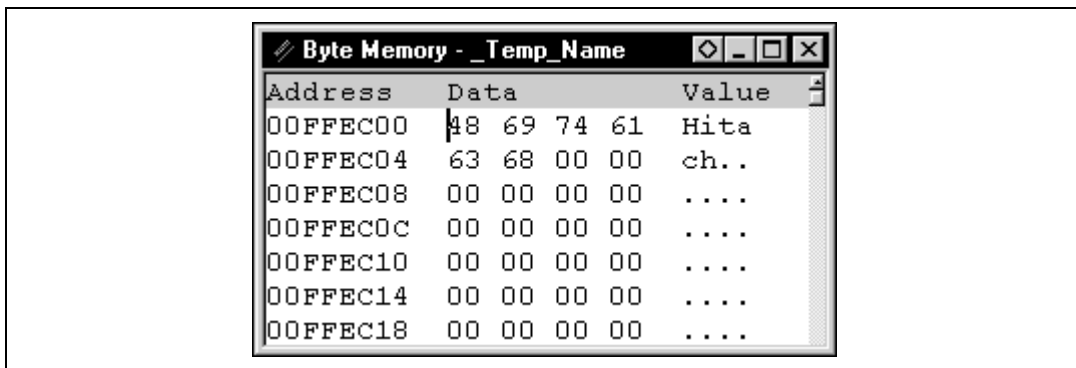
### 5.7.2 Edit Value...

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

### 5.7.3 Radix

Changes the radix for the selected local variable display.

## 5.8 Memory Window



**Figure 5.14 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, long word, 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 may be edited by either typing at the current cursor position, or 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:

### 5.8.1 Refresh

Updates the contents in the **Memory** window.

### 5.8.2 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.

### 5.8.3 Save...

Launches the **Save Memory As** dialog box, allowing the user to save a block of the debugging platform's memory to an S-Record file (\*.mot). The start and end fields may be set similarly to the Search option (see section 5.8.8, Search...).

### 5.8.4 Test...

This function is not supported by this simulator/debugger version.

Launches the **Test Memory** dialog box, allowing the user to test a block of memory within the debugging platform. The details of the test depend on the debugging platform. The start and end fields may be set similarly to the Search option(see section 5.8.8, Search...).

### 5.8.5 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 5.8.8, Search...).

### 5.8.6 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 5.8.8, Search...).

### 5.8.7 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.



### 5.8.8 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.

### 5.8.9 Set Address...

Displays the **Set Address** dialog box. When a new start address is entered, the window is updated, and the address entered by the user is shown in the upper left corner. When an overloaded function or a class name including a member function is entered for the address, the **Select Function** dialog box is displayed. In this dialog box, select the functions to be set.

### 5.8.10 ASCII/Byte/Word/Long/Single Float/Double Float

A check mark next to these six options indicates the current view format. The user may select a different option to change to that format.

## 5.9 Performance Analysis Window

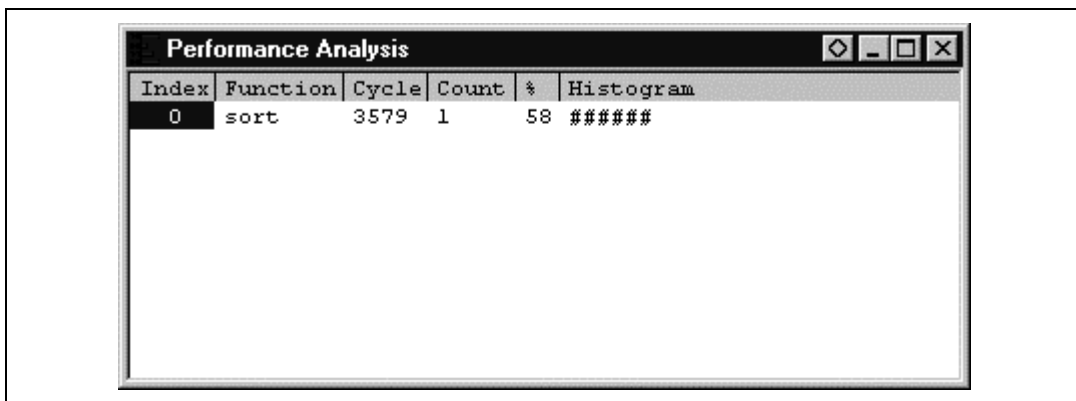


Figure 5.15 Performance Analysis Window

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 required 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. Up to 255 functions can be specified.

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

### 5.9.1 Add Range...

Launches the **Performance Option** dialog box, allowing the user to add a new function to be evaluated.

### 5.9.2 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.

### 5.9.3 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.

### 5.9.4 Reset Counts/Times

Clears the current performance analysis data.

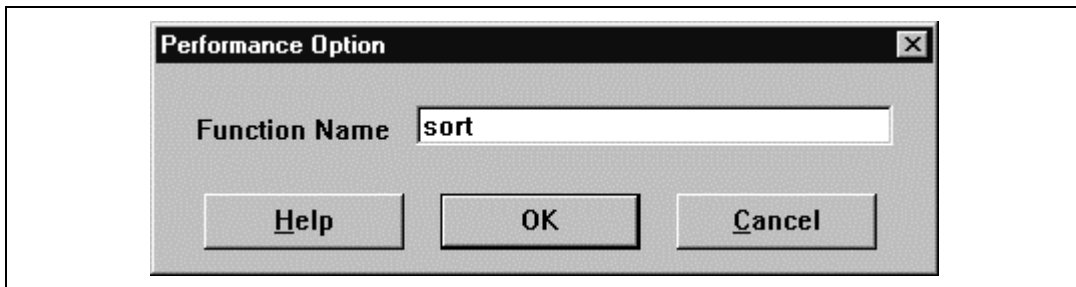
### 5.9.5 Delete All Ranges

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

### 5.9.6 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 Performance Option Dialog Box



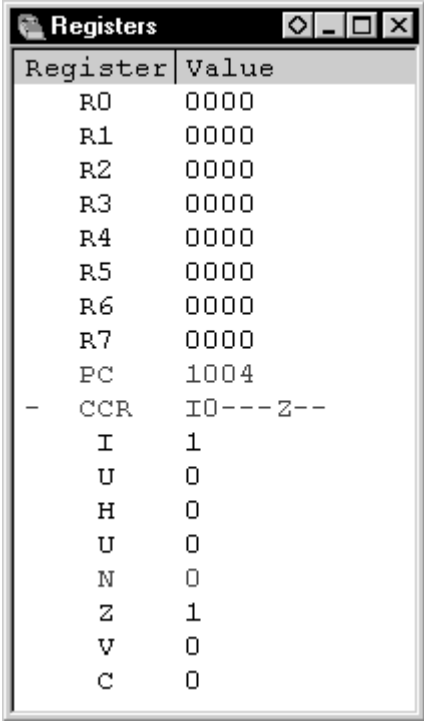
**Figure 5.16 Performance Option Dialog Box**

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 14, Selecting Functions.

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

## 5.11 Registers Window

A screenshot of a software window titled "Registers". The window contains a table with two columns: "Register" and "Value". The registers listed are R0 through R7, PC, CCR, I, U, H, U, N, Z, V, and C. The values for R0-R7 are 0000, PC is 1004, CCR is 10---Z--, I is 1, and U, H, U, N, Z, V, C are 0.

Register	Value
R0	0000
R1	0000
R2	0000
R3	0000
R4	0000
R5	0000
R6	0000
R7	0000
PC	1004
- CCR	10---Z--
I	1
U	0
H	0
U	0
N	0
Z	1
V	0
C	0

**Figure 5.17 Registers 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:

### 5.11.1 Copy



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

### 5.11.2 Edit...

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

### 5.11.3 Toggle Bit

Only available if the text cursor is placed on a bit-field, e.g. a flag within a status register. Changes the current state of the bit to its other state, e.g. a set overflow flag can be cleared.

## 5.12 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.

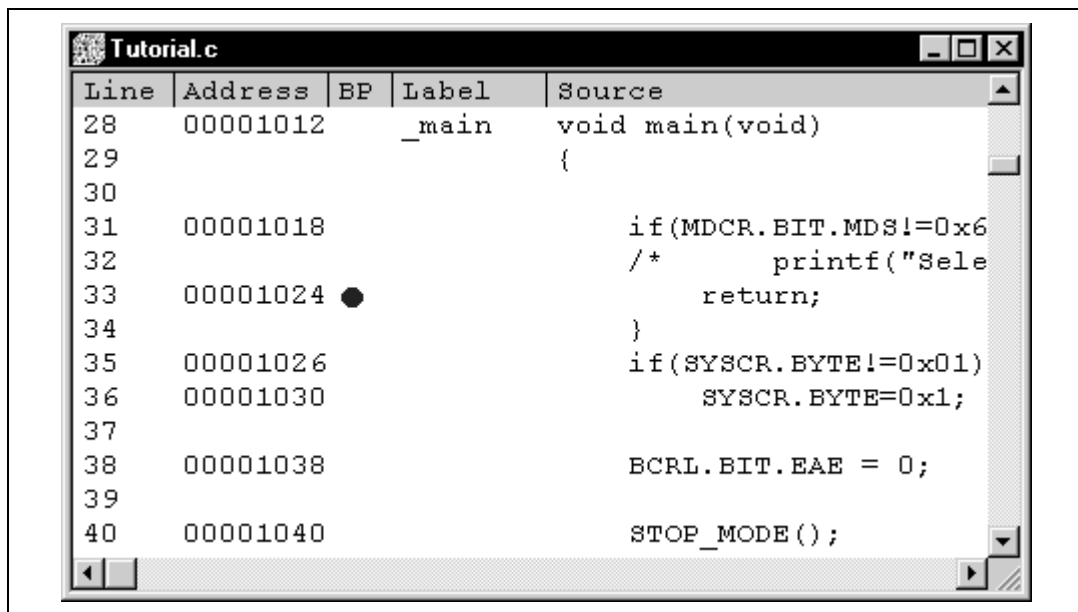


Figure 5.18 Source View

It supports column-specific double-click actions:

- **BP** - Sets/clears a program (PC) breakpoint at that address.
- **Address** - Launches the **Set Address** dialog box, allowing the user to enter a new address. If the address is within the range of this file, then the view will scroll such that the cursor can be positioned correctly. If the address is in a different source file, then that file will be opened in a new window with the cursor set to the specified address. Finally, if the address does not correspond to a source file, then a new **Disassembly** window will be opened. When an overloaded function or a class name is entered, the **Select Function** dialog box opens for you to select a function.
- **Label** - Launches the **Label** dialog box, allowing the user to enter a new label and edit the name of an existing label.

- **Line** - Launches the **Set Line** dialog box, allowing the user to go directly to a line in the source file.
- **Source** - Opens the source file in the editor (specified in the Startup menu HDI shortcut) at this source line.

Within the BP column a list of currently supported standard breakpoint types can be displayed by right clicking. The currently selected standard breakpoint is shown by a check mark to the left of the menu text.

A popup menu containing the following options is available by right clicking in any of the other columns within the window:

### 5.12.1 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.

### 5.12.2 Find...



Launches the **Find** dialog box, allowing the user to search the source file for a string.

### 5.12.3 Set Address...

Launches the **Set Address** dialog box, allowing the user to enter a new start address. 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 name is entered, the **Select Function** dialog box opens for you to select a function.

### 5.12.4 Set Line...

Launches the **Set Line** dialog box, allowing the user to display and move the text cursor (not the mouse cursor) to a specific line.

### 5.12.5 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. Grayed if not supported by the debugging platform.

### 5.12.6 Set PC Here

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

### 5.12.7 Instant Watch...

Launches the **Instant Watch** dialog box with the name extracted from the view at the current text cursor (not mouse cursor) position. Only valid in the source column.

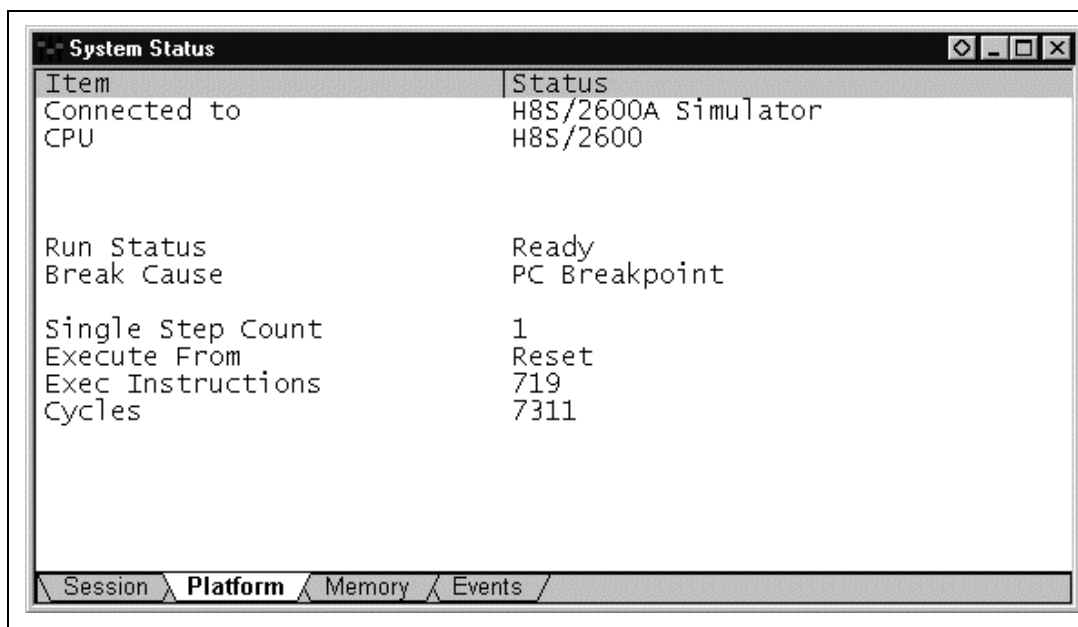
### 5.12.8 Add Watch

Adds the name extracted from the view at the current text cursor (not mouse cursor) position to the list of watched variables. If the **Watch** window is not open, then it is opened and brought to the top of the child windows. Only valid in the source column.

### 5.12.9 Go To Disassembly

Opens a Disassembly view at the address matching the current source line.

## 5.13 System Status Window



**Figure 5.19 System Status Window**

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

The **System Status** window is split into four tabs:

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

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

### 5.13.1 Update

Updates the displayed data.

### 5.13.2 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.

## 5.14 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.

PTR	CYCLE	ADDR	CCR	MULT	INSTRUCTION	Source
-0708	0000000017	000000	I----	Z---	MOV.L ER6,@ER7 00003FFC<-00000000	void main(void)
-0707	0000000021	000004	I-----	---	MOV.L ER7,ER6 ER6<-00003FFC	
-0706	0000000033	000006	I-----	---	SUB.L #H'00000032,ER7 ER7<-00003FCA	
-0705	0000000037	00000C	I----	Z---	SUB.W RO,RO RO<-0000	for( i=0; i<10; i++ ){
-0704	0000000049	00000E	I----	Z---	MOV.W RO,@(H'FFD2:16,ER6) 00003FCE<-0000	
-0703	0000000057	000012	I----	Z---	BRA @H'005C:8 PC<-0000005C	
-0702	0000000069	00005C	I----	Z---	MOV.W @(H'FFD2:16,ER6),RO RO<-0000	
-0701	0000000077	000060	I-H-N--	C---	CMP.W #H'000A,RO	
-0700	0000000085	000064	I-H-N--	C---	BLT @H'0014:8 PC<-00000014	
-0699	0000000102	000014	I-H-N--	C---	JSR @_rand:24 PC<-000002A6	j = rand();
-0698	0000000119	0002A6	I-H----	C---	MOV.L ER6,@ER7 00003FCE<-00003FFC	
-0697	0000000131	0002AA	I-H----	C---	MOV.L #H'0000035A,ER6 ER6<-0000035A	
-0696	0000000147	0002B0	I-H--Z-	C---	MOV.L @ER6,ERO ERO<-00000000	

Figure 5.20 Trace Window



This window displays the following trace information items. The trace acquisition conditions can be specified in the **Trace Acquisition** dialog box.

[PTR]	Pointer in the trace buffer (0 for the last executed instruction)
[CYCLE]	Total number of instruction execution cycles (cleared by pipeline reset)
[ADDR]	Instruction address
[CCR]	Contents of the condition code register (CCR) are displayed in mnemonics
[MULT]	Internal flags in the multiplier are displayed in mnemonics (only for the H8S/2600 series)
[INSTRUCTION]	Instruction mnemonic and data access (displayed in the form of [Transfer destination <- Transferred 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:

#### 5.14.1 Find...

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

#### 5.14.2 Find Next

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

#### 5.14.3 Filter...

This function is not supported by this simulator/debugger version.

Launches the **Filter Trace** dialog box, allowing the user to mask out all unnecessary trace entries.

#### 5.14.4 Acquisition...

Launches the **Trace Acquisition** dialog box, allowing the user to define the area of user program to be traced. This is useful to focus tracing on problem areas.

#### 5.14.5 Halt

This function is not supported by this simulator/debugger version.

Stops tracing data and updates the trace information without stopping execution of the user program.

#### 5.14.6 Restart

This function is not supported by this simulator/debugger version.

Starts tracing data.

#### 5.14.7 Snapshot

This function is not supported by this simulator/debugger version.

Updates the trace information to show the debugging platform's current status without stopping user program execution.

#### 5.14.8 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.

#### 5.14.9 Save...

Launches the **Save As** file dialog box, allowing the user to save the contents of the trace buffer as a text file. It is possible to define a numeric range based on the Cycle number or to save the complete buffer (saving the complete buffer may take several minutes). Note that this file cannot be reloaded into the trace buffer.

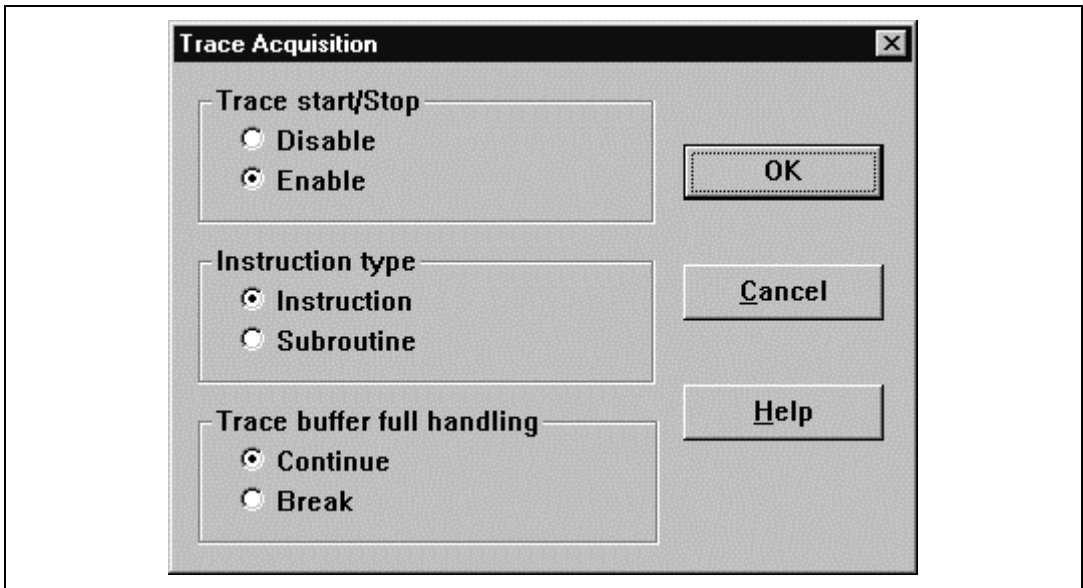
#### 5.14.10 View Source

Opens a **Source** or **Disassembly** window for the address.

#### 5.14.11 Trim Source

Removes white space from the left side of the source.

## 5.15 Trace Acquisition Dialog Box



**Figure 5.21 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.

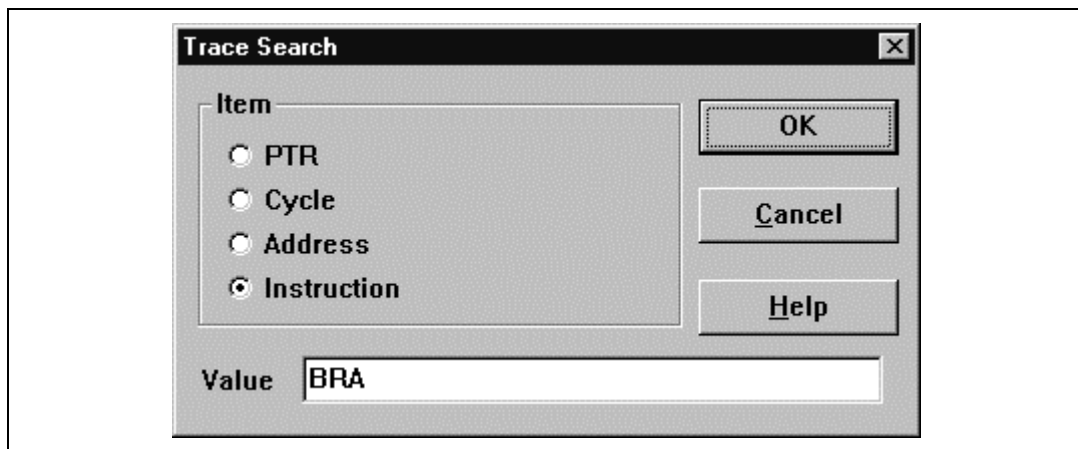
[Trace buffer full handling]

[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.

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

## 5.16 Trace Search Dialog Box



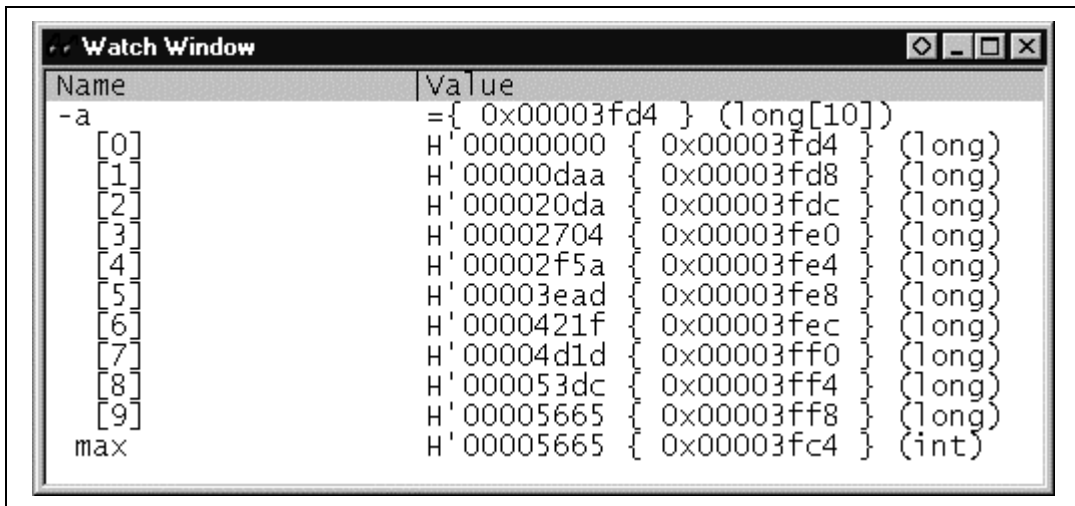
**Figure 5.22 Trace Search Dialog Box**

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.

## 5.17 Watch Window



**Figure 5.23 Watch Window**

Allows the user to view and modify C/C++-source level variables. The contents of this window are blank unless the current user program can be associated to a C/C++-source file *via* the debugging information available in the absolute file (\*.abs).

[Name] Variable name

[Value] Value, allocated position, and type of the 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 reduced. Alternatively, the plus and minus keys may be used.

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

### 5.17.1 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.

### 5.17.2 Delete

Removes the variable indicated by the text cursor (not the mouse cursor) from the **Watch** window.

### 5.17.3 Delete All

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

### 5.17.4 Add Watch...

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

### 5.17.5 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.

### 5.17.6 Radix

Modifies the radix for the selected watch item display.

## 5.18 System Configuration Dialog Box

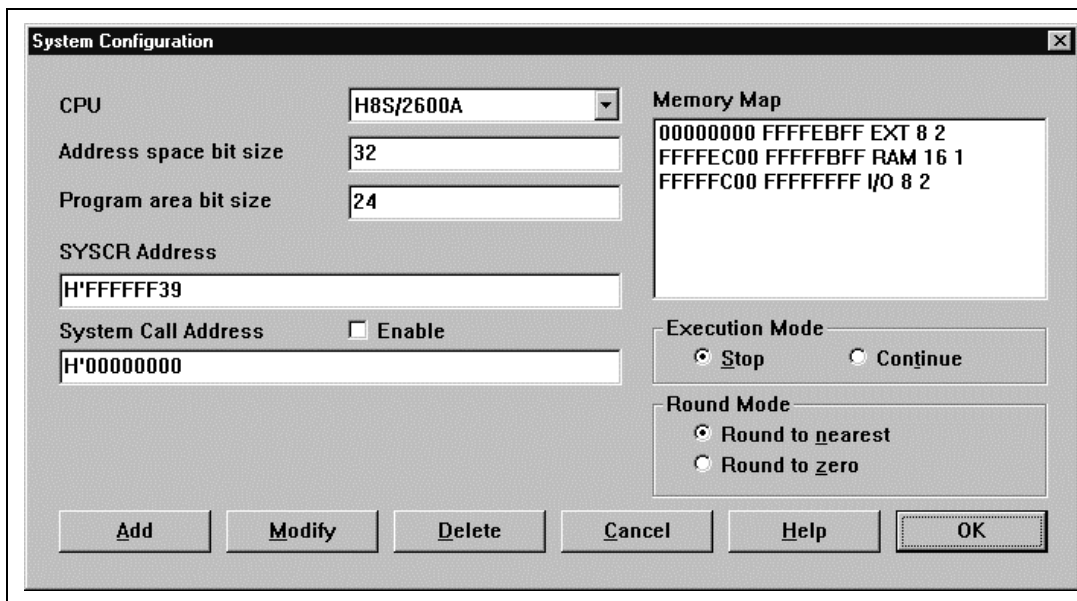


Figure 5.24 System Configuration Dialog Box

This dialog box sets the address space bit size, program area bit size, SYSCR address, system call start address, execution mode, floating-point rounding mode, and memory map.

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

[Address Space Bit Size]

Specifies the bit size of the address space. Available bit size depends on the CPU as follows:

H8/300, H8/300L, H8/300HN, H8S/2600N, H8S/2000N: 16 bits

H8/300HA: 17 to 24 bits

H8S/2600A, H8S/2000A: 17 to 32 bits

[Program Area Bit Size]

Specifies the bit size of the program area. Available bit size depends on the CPU as follows:

H8/300, H8/300L, H8/300HN, H8S/2600N, H8S/2000N: 16 bits

H8/300HA: 17 to 24 bits (same as the address space bit size)

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

[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. The memory types are as follows:

[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** 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 this button.

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

## 5.19 Memory Map Modify Dialog Box

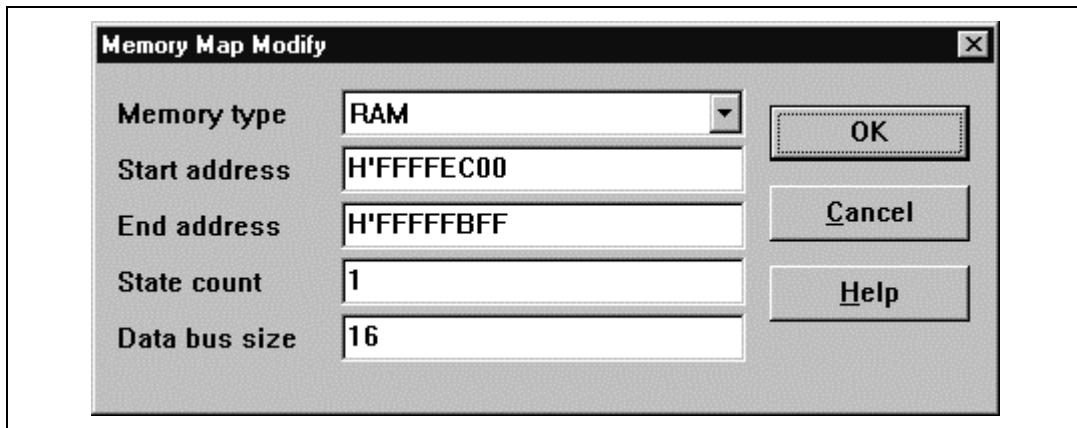


Figure 5.25 Memory Map Modify Dialog Box

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 Memory Map dialog box, then delete or modify the memory map setting.



## 5.20 Memory Map Dialog Box

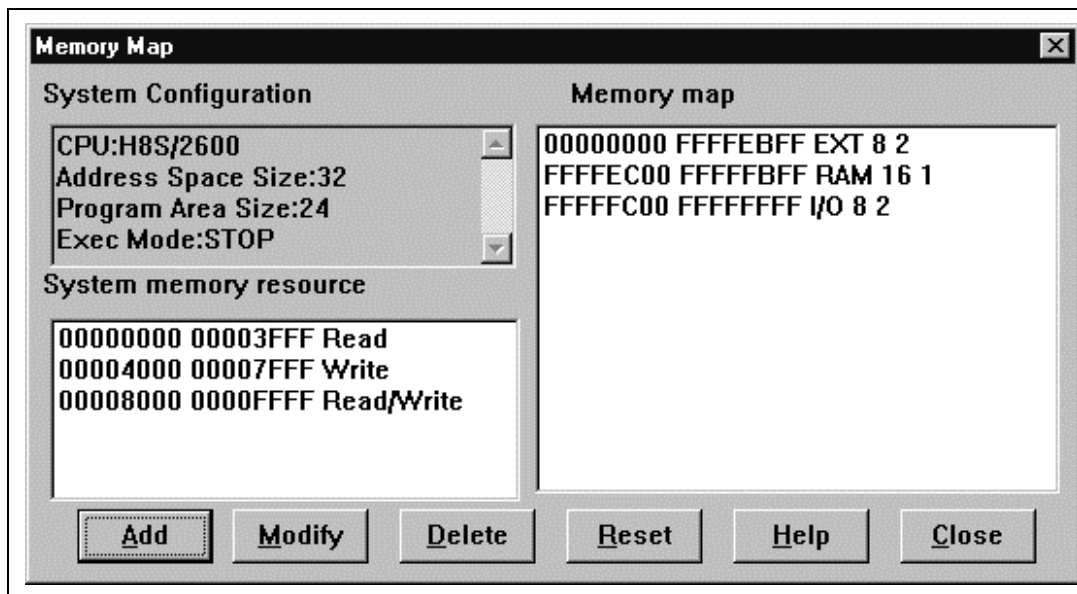


Figure 5.26 Memory Map Dialog Box

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

[System Configuration] Displays the target CPU, address space size, program area size, and execution mode of the simulator/debugger.

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

[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 the **Modify** 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.

## 5.21 System Memory Resource Modify Dialog Box

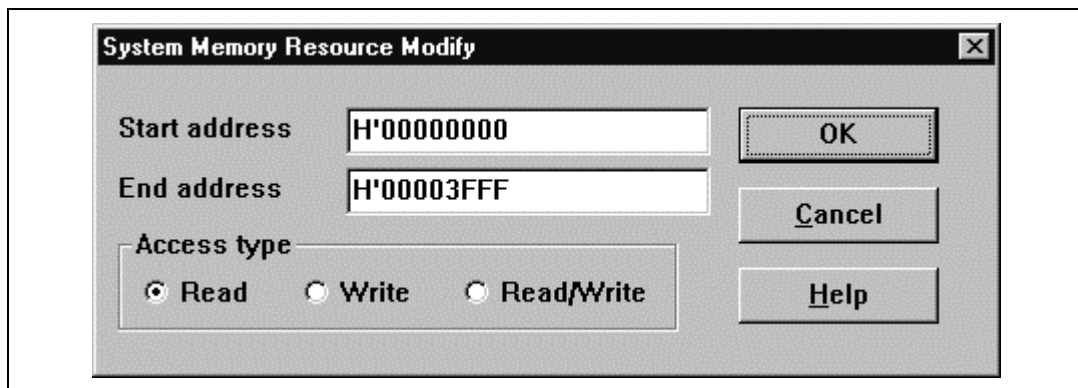


Figure 5.27 System Memory Resource Modify Dialog Box

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.

## 5.22 Control Registers Window



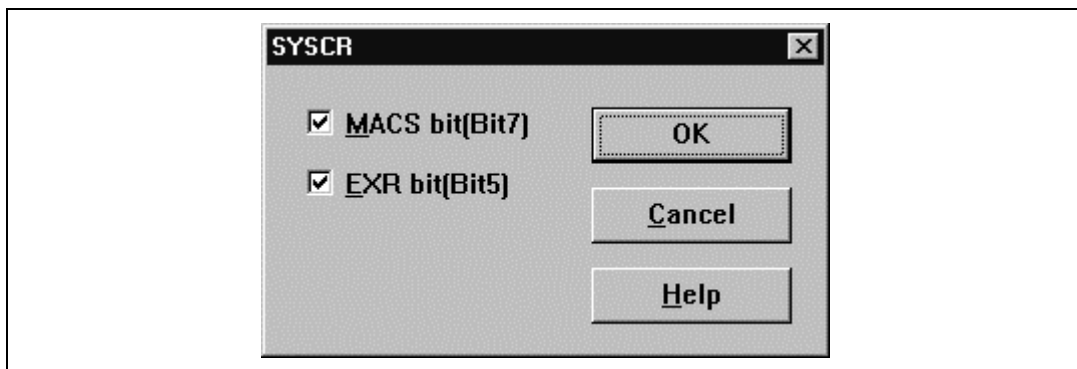
**Figure 5.28 Control Registers Window**

This window displays the following control register value. Note that this window is available only for the H8S/2600 series.

[SYSCR] System control register

The control register value can be directly modified in the window. Double-clicking the register opens the SYSCR dialog box. In this dialog box, the register value can be modified in bit units.

## 5.23 SYSCR Dialog Box



**Figure 5.29 SYSCR Dialog Box**

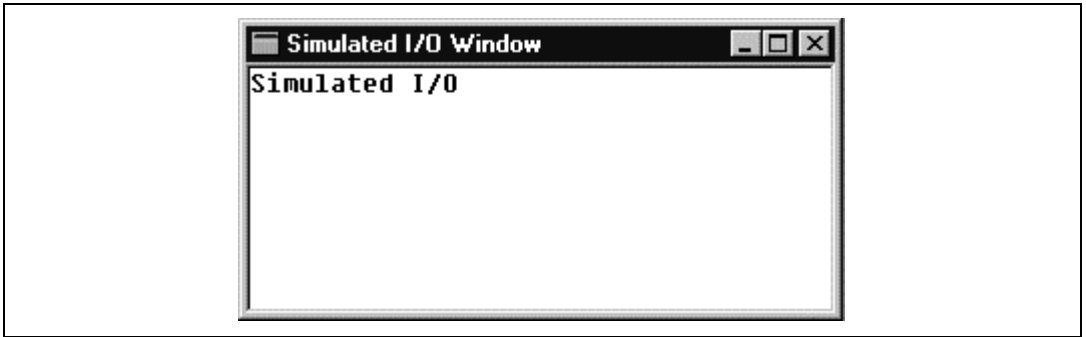
This dialog box sets the value of the SYSCR register (system control register). Note that this dialog box is available only for the H8S/2600 series.

The following items are set:

- [MACS bit] Selects saturation operation or non-saturation operation for the multiply and accumulate operation (MAC instruction). Checking this item selects the saturation operation.
- [EXR bit] Enables or disables the EXR register. Checking this item enables the EXR register.

Clicking the **[OK]** button sets modified values in the memory. Clicking the **[Cancel]** button closes this dialog box.

## 5.24 Simulated I/O Window



**Figure 5.30 Simulated I/O Window**

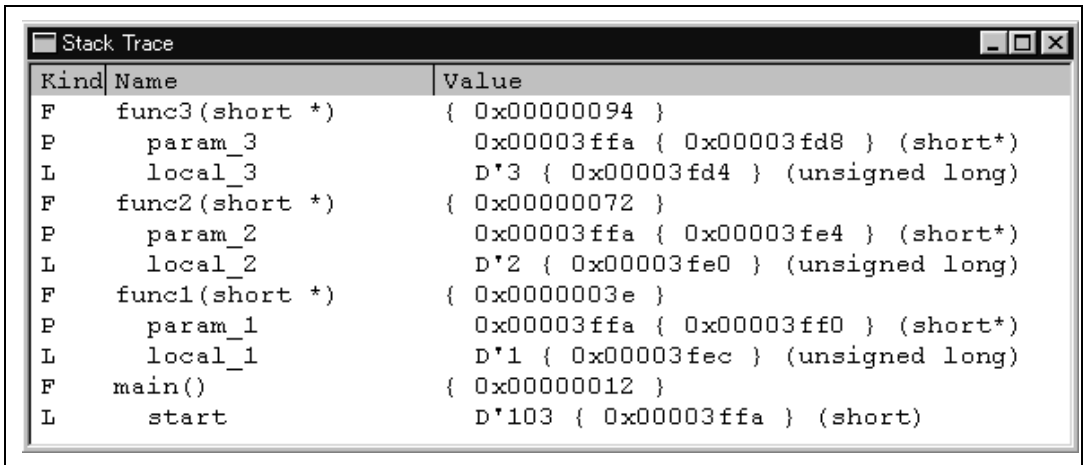
This window is for standard I/O and file I/O system calls from the user program.

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 3.12, Standard I/O and File I/O Processing.

## 5.25 Stack Trace Window



Kind	Name	Value
F	func3(short *)	{ 0x00000094 }
P	param_3	0x00003ffa { 0x00003fd8 } (short*)
L	local_3	D*3 { 0x00003fd4 } (unsigned long)
F	func2(short *)	{ 0x00000072 }
P	param_2	0x00003ffa { 0x00003fe4 } (short*)
L	local_2	D*2 { 0x00003fe0 } (unsigned long)
F	func1(short *)	{ 0x0000003e }
P	param_1	0x00003ffa { 0x00003ff0 } (short*)
L	local_1	D*1 { 0x00003fec } (unsigned long)
F	main()	{ 0x00000012 }
L	start	D*103 { 0x00003ffa } (short)

**Figure 5.31 Stack Trace Window**

This window displays the function call history except for the functions called after an interrupt.

The following items are displayed.

[Kind]                    Symbol type  
F: Function  
P: Parameter of a function  
L: Local variable

[Name]                    Symbol name

[Value]                   Symbol value, address, and type

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

### 5.25.1 Copy

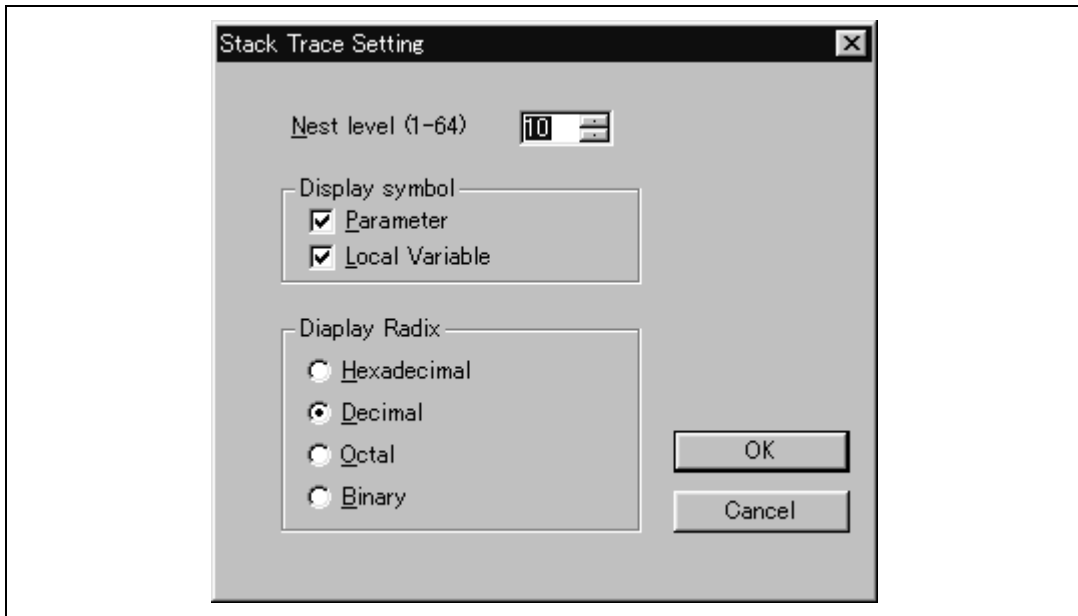
This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

### 5.25.2 Go to Source

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

### 5.25.3 View Setting...

Launches the **Stack Trace Setting...** dialog box allowing the user to specify the display format of the **Stack Trace** window.



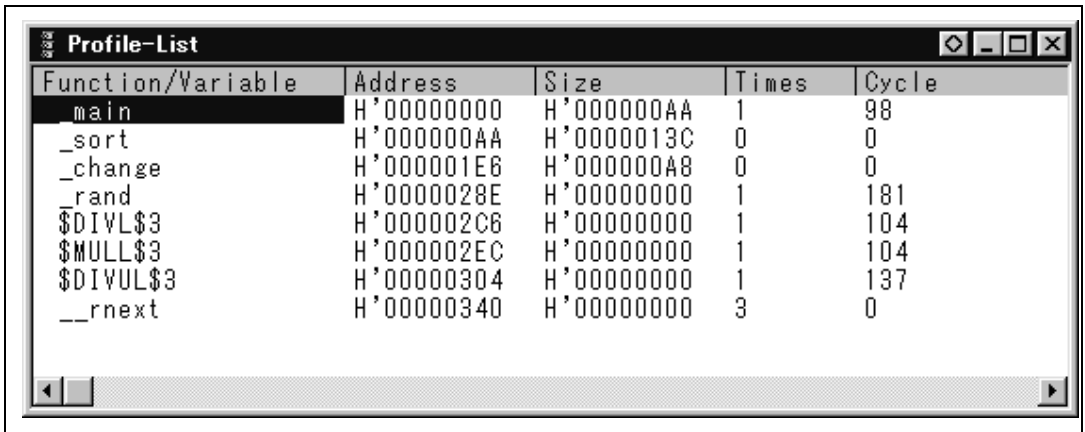
**Figure 5.32 Stack Trace Setting Dialog Box**

**[Nest level]** specifies the nesting level of a function call to be displayed in the **Stack Trace** window. The display symbol group check box specifies the symbol (excluding the functions) to be displayed. Display Radix group check box specifies the display radix in the **Stack Trace** window.

**[Display symbol]** group check boxes specify the symbol types to be displayed in addition to functions.

**[Display Radix]** group radio buttons specify the radix for displays in the **Stack Trace** window.

## 5.26 Profile-List Window



The screenshot shows a window titled "Profile-List" with a table containing the following data:

Function/Variable	Address	Size	Times	Cycle
<b>_main</b>	H'00000000	H'000000AA	1	98
_sort	H'000000AA	H'0000013C	0	0
_change	H'000001E6	H'000000A8	0	0
_rand	H'0000028E	H'00000000	1	181
\$DIVL\$3	H'000002C6	H'00000000	1	104
\$MULL\$3	H'000002EC	H'00000000	1	104
\$DIVUL\$3	H'00000304	H'00000000	1	137
__rnext	H'00000340	H'00000000	3	0

**Figure 5.33 Profile-List Window**

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. Displayed profile data is as follows:

- Called (the number of times a global variable is accessed)
- Cycle (the number of execution cycles)

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

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

Double-clicking the Function/Variable or Address column displays the source program or disassembled memory contents corresponding to the address in the line. Right-clicking on the mouse within the window displays a popup menu. Supported menu options are described in the following sections:

### 5.26.1 View Source

Displays the source program or disassembled memory contents for the address in the selected line. If a line of a global variable is selected, this menu option is displayed in gray characters.

### 5.26.2 View Profile-Tree

Displays the **Profile-Tree** window.

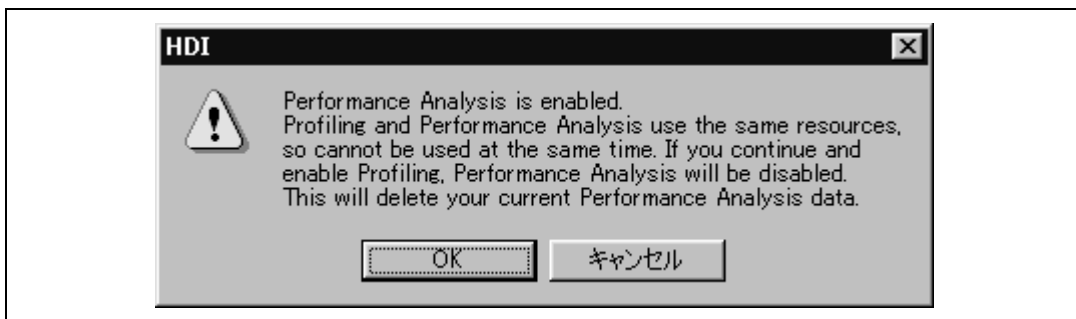


### 5.26.3 View Profile-Chart

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

### 5.26.4 Enable Profiler

Toggles the collection of profile data. When profile data acquisition is active, a check mark is shown to the left of the menu text. Profile data and performance analysis data cannot be acquired at the same time. If the profile data acquisition is going to be enabled when the performance analysis data acquisition is active (when the “Enable Analysis” in the **Performance Analysis** window is checked), a warning message box is displayed.



**Figure 5.34 Warning Message Box Showing Profiler and Analysis Cannot Be Set at a Time**

When [OK] is clicked, the performance analysis data acquisition is disabled and the profile data acquisition is enabled.

### 5.26.5 Find...

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

### 5.26.6 Clear Data

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

### 5.26.7 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 optimizing linkage editor's manual supplied in a separate volume.

### 5.26.8 Output Text File...

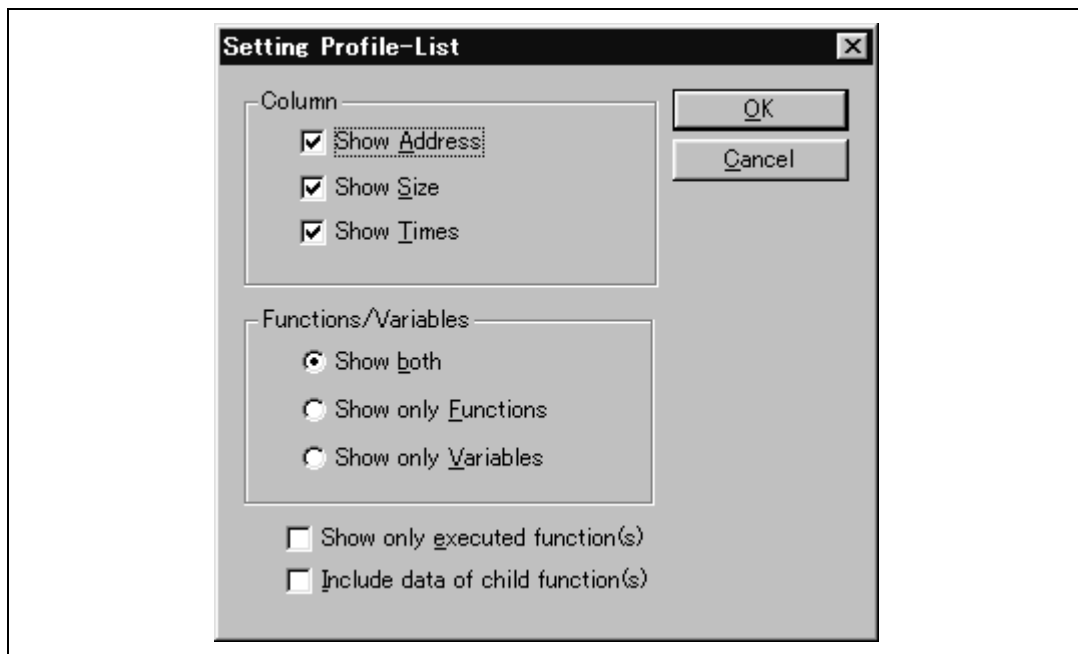
Displays the **Save Text of Profile Data** dialog box. Displayed contents are saved in a text file.

### 5.26.9 Select Data...

Selects profile data types. The types of profile data differ according to the debug platform. If this menu option is not supported by the debug platform, it is displayed in gray characters.

### 5.26.10 Setting...

Displays the **Setting Profile-List** dialog box to set displayed contents.



**Figure 5.35 Setting Profile-List Dialog Box**

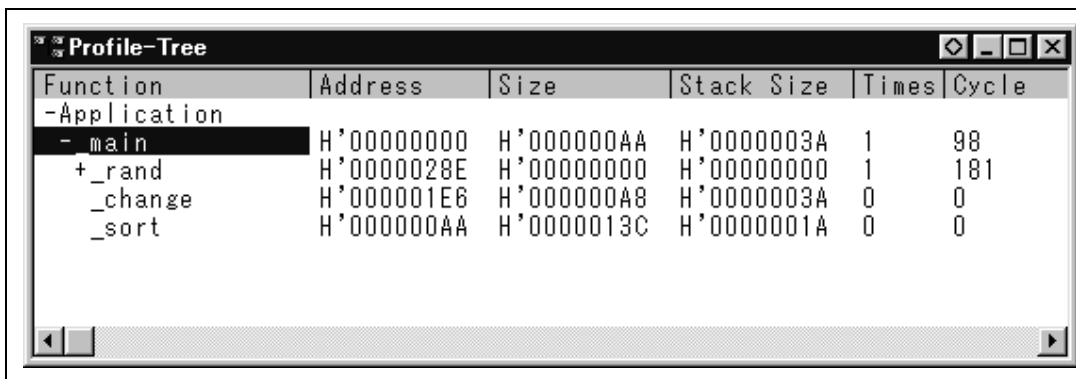
The Column group check boxes set the display of a specific column.

Functions/Variables group radio buttons are set to display either both of the functions and the global variables displayed in the Function/Variable column or only one of them.

Checking in the Show Only Executed Function(s) check box disables the display of unexecuted functions. If a stack usage information file (.sni extension) output from the optimizing linkage editor does not exist, unexecuted functions are not displayed even if this check box is not checked.

The Include Data of Child Function(s) check box sets whether to display information for a child function which is called in the function as profile data.

## 5.27 Profile-Tree Window



Function	Address	Size	Stack Size	Times	Cycle
-Application					
- main	H'00000000	H'000000AA	H'0000003A	1	98
+ _rand	H'0000028E	H'00000000	H'00000000	1	181
_change	H'000001E6	H'000000A8	H'0000003A	0	0
_sort	H'000000AA	H'0000013C	H'0000001A	0	0

**Figure 5.36 Profile-Tree Window**

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 displayed each time a function call occurs.

Displayed profile data is as follows:

- Cycle (the number of execution cycles)

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 usage information file (.sni extension) output from the optimizing linkage editor, the stack size is not displayed. For details of the stack usage information file, refer to the optimizing linkage editor's manual.

Double-clicking a function in the Function column expands or reduces the tree structure display. The expansion or reduction is also provided by the plus and minus keys. Double-clicking the

Address column displays the source program or disassembled memory 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:

### 5.27.1 View Source

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

### 5.27.2 View Profile-List

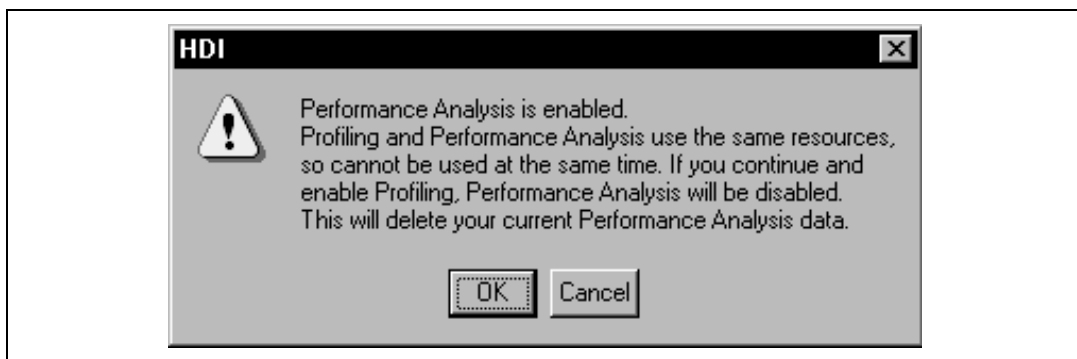
Displays the **Profile-List** window.

### 5.27.3 View Profile-Chart

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

### 5.27.4 Enable Profiler

Toggles the collection of profile data. When profile data acquisition is active, a check mark is shown to the left of the text. Profile data and performance analysis data cannot be acquired at the same time. If the profile data acquisition is going to be enabled when the performance analysis data acquisition is active (when the “Enable Analysis” in the **Performance Analysis** window is checked), a warning message box is displayed.



**Figure 5.37 Warning Message Box Showing Profiler and Analysis Cannot Be Set at a Time**

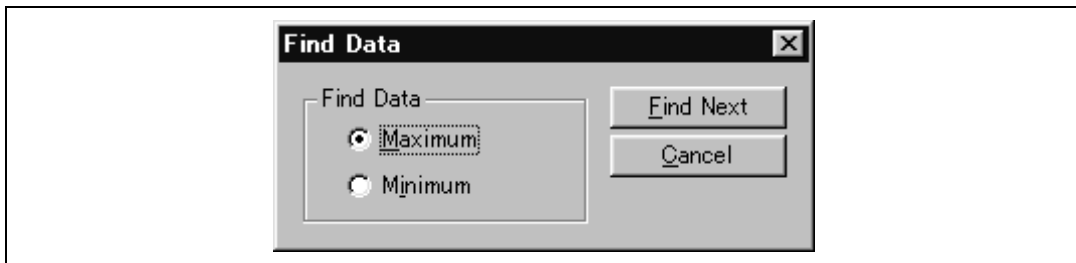
When [OK] is clicked, the performance analysis data acquisition is disabled and the profile data acquisition is enabled.

### 5.27.5 Find...

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

### 5.27.6 Find Data...

Displays the **Find Data** dialog box. When the cursor is in the Function column, this menu option is displayed in gray characters.



**Figure 5-38 Find Data Dialog Box**

By selecting the search type from 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.

### 5.27.7 Clear Data

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

### 5.27.8 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 optimizing linkage editor's manual.

### 5.27.9 Output Text File...

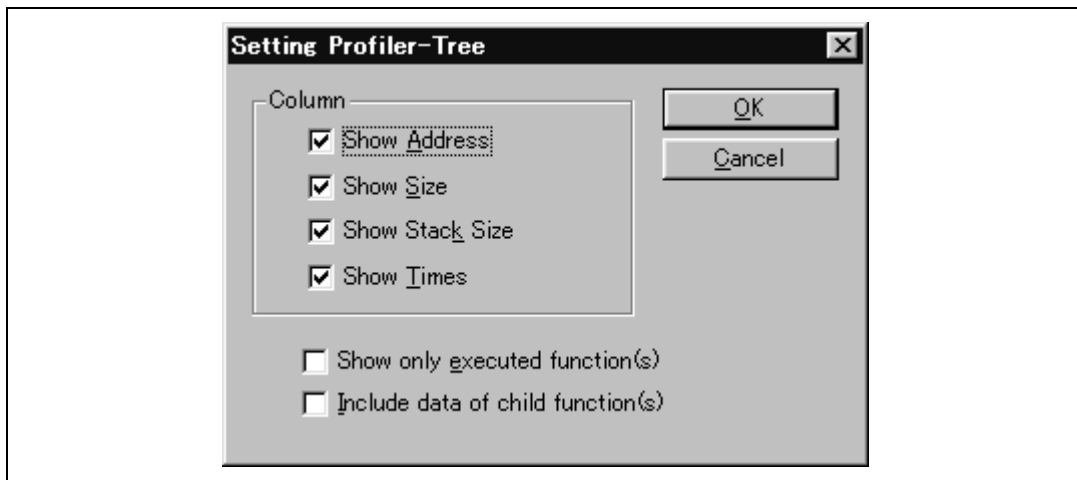
Displays the **Save Text Profile Data** dialog box. Displayed contents are saved in a text file.

## 5.27.10 Select Data...

Selects profile data types. The types of profile data differ according to the debug platform. If this menu option is not supported by the debug platform, it is displayed in gray characters.

## 5.27.11 Setting...

Displays the **Setting Profile-Tree** dialog box to set displayed contents.



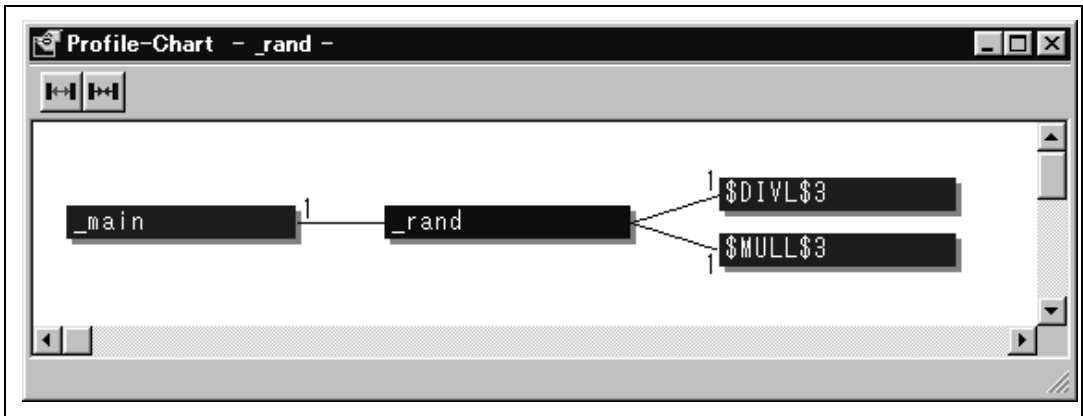
**Figure 5.39 Setting Profile-Tree Dialog Box**

The Column group check boxes set whether or not to display a specific column.

Checking in the Show Only Executed Function(s) check box disables displaying unexecuted functions. If a stack usage information file (.sni extension) output from the optimizing linkage editor does not exist, unexecuted functions are not displayed even if this check box is not checked.

The Include Data of Child Function(s) check box sets whether to display information for a child function called in a function as profile data.

## 5.28 Profile-Chart Window



**Figure 5.40 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 **Profile-List** window or **Profile Tree** 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.

The **Profile-Chart** window includes the following tool buttons:

- Expands Size
- Reduces Size

Right-clicking on the mouse within the window displays a popup menu. Supported menu options are described in the section 5.28.3, View Source and in the subsequent sections.

### 5.28.1 Expands Size



Expands spaces between each function. The plus key can also be used to expand spaces.

### 5.28.2 Reduces Size



Reduces spaces between each function. The minus key can also be used to reduce spaces.

### 5.28.3 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.

### 5.28.4 View Profile-List

Displays the **Profile-List** window.

### 5.28.5 View Profile-Tree

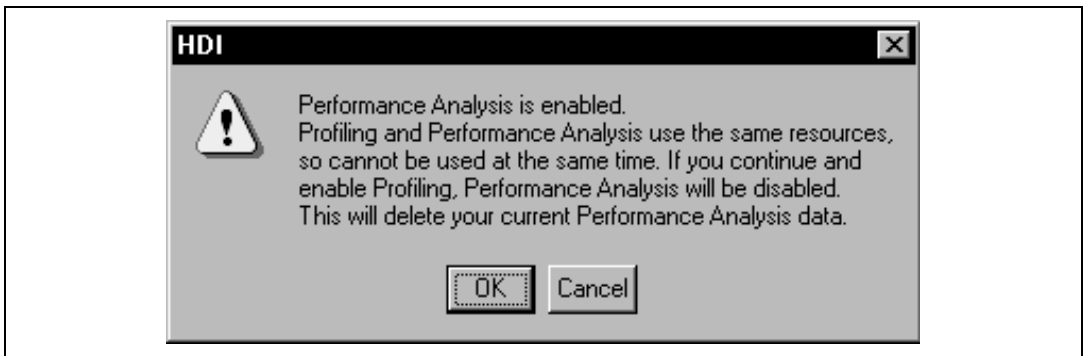
Displays the **Profile-Tree** window.

### 5.28.6 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.

### 5.28.7 Enable Profiler

Toggles the collection of profile data. When profile data acquisition is active, a check mark is shown to the left of the text. Profile data and performance analysis data cannot be acquired at the same time. If the profile data acquisition is enabled when the performance analysis data acquisition is active (when the Enable Analysis check box in the **Performance Analysis** window is checked), a warning message box is displayed.



**Figure 5.41 Warning Message Box Showing Profiler and Analysis Cannot Be Set at a Time**



When [OK] is clicked, the performance analysis data acquisition is disabled and the profile data acquisition is enabled.

### 5.28.8 Clear Data

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

### 5.28.9 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.

### 5.28.10 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 optimizing linkage editor's manual.



# Section 6 Command Lines

Table 6.1 lists the commands.

**Table 6.1 Simulator/Debugger Commands**

Command Name	Abbreviation	Function
!	-	Comment
ANALYSIS	AN	Enables or disables performance analysis
ANALYSIS_RANGE	AR	Sets or displays performance analysis functions
ANALYSIS_RANGE_DELETE	AD	Deletes a performance analysis range
ASSEMBLE	AS	Assembles instructions into memory
ASSERT	-	Checks if an expression is true or false
BREAKPOINT	BP	Sets a breakpoint at an instruction address
BREAK_ACCESS	BA	Specifies a memory range access as a break condition
BREAK_CLEAR	BC	Deletes breakpoints
BREAK_DATA	BD	Specifies a memory data value as a break condition
BREAK_DISPLAY	BI	Displays a list of breakpoints
BREAK_ENABLE	BE	Enables or disables a breakpoint
BREAK_REGISTER	BR	Specifies a register data as a break condition
BREAK_SEQUENCE	BS	Sets sequential breakpoints
DISASSEMBLE	DA	Disassembles memory contents
ERASE	ER	Clears the <b>Command Line</b> window
EVALUATE	EV	Evaluates an expression
FILE_LOAD	FL	Loads an object (program) file
FILE_SAVE	FS	Saves memory to a file
FILE_VERIFY	FV	Verifies file contents against memory
GO	GO	Executes user program
GO_RESET	GR	Executes user program from reset
GO_TILL	GT	Executes user program until temporary breakpoint
HALT	HA	Halts user program
HELP	HE	Gets help for command line or help on a command
INITIALISE	IN	Initializes HDI

**Table 6.1 Simulator/Debugger Commands (cont)**

<b>Command Name</b>	<b>Abbreviation</b>	<b>Function</b>
LOG	LO	Controls command output logging
MAP_DISPLAY	MA	Displays memory mapping
MAP_SET	MS	Allocates a memory area
MEMORY_DISPLAY	MD	Displays memory contents
MEMORY_EDIT	ME	Modifies memory contents
MEMORY_FILL	MF	Fills a memory area
MEMORY_MOVE	MV	Moves a block of memory
MEMORY_TEST	MT	Tests a block of memory
QUIT	QU	Exits HDI
RADIX	RA	Sets default input radix
REGISTER_DISPLAY	RD	Displays CPU register values
REGISTER_SET	RS	Changes CPU register contents
RESET	RE	Resets CPU
SLEEP	-	Delays command execution
STEP	ST	Steps program (by instructions or source lines)
STEP_OUT	SP	Steps out of the current function
STEP_OVER	SO	Steps program, not stepping into functions
STEP_RATE	SR	Sets rate of stepping
SUBMIT	SU	Executes a command file
SYMBOL_ADD	SA	Defines a symbol
SYMBOL_CLEAR	SC	Deletes a symbol
SYMBOL_LOAD	SL	Loads a symbol information file
SYMBOL_SAVE	SS	Saves a symbol information file
SYMBOL_VIEW	SV	Displays symbols
TRACE	TR	Displays trace buffer contents
TRACE_ACQUISITION	TA	Enables or disables trace information acquisition

The following describes each command syntax.

## !(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).

## 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.

## 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>	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.

## 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 [&lt;index&gt;]

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.

**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 &lt;address&gt;

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

**Example:**

AS H'1000      Starts assembling from H'1000.

**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.

## BREAKPOINT

### Abbreviation: BP

### Description:

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

### Syntax:

bp <address> [<count>]

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 (optional, default = 1).

### 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 A break occurs when an attempt is made to execute the instruction at address H'C0.

## BREAK\_ACCESS

### Abbreviation: BA

### Description:

Specifies a memory range as a break condition



## Syntax:

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

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.

## 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.

## 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.

## BREAK\_DATA

**Abbreviation: BD**

**Description:**

Specifies a memory data value as a break condition.

**Syntax:**

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

Parameter	Type	Description
<address>	Numeric	The address where the break condition is checked.
<data>	Numeric	Access data
<size>	Keyword	Size (optional, default = L).
	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.

**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 1000	A break occurs when H'1000 is written to memory address H'4000 in longword.

## 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.

**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.

# BREAK\_REGISTER

**Abbreviation:** BR

**Description:**

Specifies a register data as a break condition

**Syntax:**

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

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.

**Examples:**

BREAK\_REGISTER R0 FFFF W EQ      A break occurs when the low-order two bytes of the R0 register change to H'FFFF.

BR R10      A break occurs when the R10 register is written to.

## BREAK\_SEQUENCE

**Abbreviation:** BS

**Description:**

Sets sequential breakpoints

**Syntax:**

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

Parameter	Type	Description
<address1> - <address8>	Numeric	Addresses of sequential breakpoints. Up to eight addresses can be specified.

**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.

## 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.

## ERASE

**Abbreviation:** ER

### Description:

Clears the **Command Line** window

### Syntax:

er

Parameter	Type	Description
none		Clears the <b>Command Line</b> window

### Example:

ER                              Clears the **Command Line** window.

## 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



## Examples:

FILE\_LOAD A:\\BINARY\\TESTFILE.A22      Loads S-Record file "testfile.a22".

FL ANOTHER.MOT H'200                      Loads Motorola S-Record file "another.mot"  
with an offset of H'200 bytes.

## FILE\_SAVE

### Abbreviation: FS

### Description:

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

### Syntax:

fs <filename> <start> <end>

Parameter	Type	Description
<filename>	String	File name
<start>	Numeric	Start address
<end>	Numeric	End address

## Examples:

FILE\_SAVE TESTFILE H'0 H'2013              Saves address range H'0-H'2013 as Motorola  
S-Record file "TESTFILE.MOT".

FS D:\\USER\\ANOTHER.A22 H'4000              Saves address range H'4000-H'4FFF as S-  
H'4FFF    Record format file "ANOTHER.A22".

## 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>	String	File name
<offset>	Numeric	Offset to be added to file address (optional, default = 0)

## Examples:

FILE\_VERIFY A:\\BINARY\\TEST.A22 Verifies 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.

## GO

### Abbreviation: GO

### Description:

Executes object code (the user program). While the user program is executing, the **Performance Analysis** window is updated.

### 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.

## 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).

## 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.

## 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.

**HELP****Abbreviation: HE****Description:**

Opens a window displaying the help file.

For context sensitive help, the F1 key should be pressed. Help on a particular command can be displayed by entering HELP or HE followed by the command name.

**Syntax:**

he [<command>]

Parameter	Type	Description
none		Displays the contents of the help
<command>	String	Displays the help for the specified command

**Examples:**

HE Displays the contents of the help.

HE GO Displays help for the GO command.

**INITIALISE****Abbreviation: IN****Description:**

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



## 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.

## 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
<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.

**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

# 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 a 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)



# 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.

## MEMORY\_MOVE

**Abbreviation:** MV

**Description:**

Moves data in the specified memory area.

**Syntax:**

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

Parameter	Type	Description
<start>	Numeric	Source start address
<end>	Numeric	Source 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.

## MEMORY\_TEST

**Abbreviation:** MT

**Description:**

Performs read, write, and verification testing in the specified address range. At this time, the original contents are destroyed. 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.

## QUIT

**Abbreviation:** QU

### Description:

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

### Syntax:

qu

Parameter	Type	Description
none		Exits HDI

### Example:

QU                                Exits HDI.

## 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.

**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

# 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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 1 is the slowest.

**Syntax:**

sr <rate>

Parameter	Type	Description
none		Displays the step rate
<rate>	Numeric	Step rate 1 to 6 (6 = fastest)



**Examples:**

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

**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. The **[stop]** button terminates the process.

**Syntax:**

su <filename>

Parameter	Type	Description
<filename>	String	File name

**Examples:**

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

**SYMBOL\_ADD****Abbreviation: SA****Description:**

Adds a symbol, or changes an existing one.

**Syntax:**

sa <symbol> <value>

Parameter	Type	Description
<symbol>	String	Symbol name
<value>	Numeric	Value

## Examples:

SYMBOL\_ADD start H'1000      Defines the symbol start at H'1000.

SA END\_OF\_TABLE 1000      Uses current default radix and defines END\_OF\_TABLE at H'1000 .

## 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>	String	Symbol name

## Examples:

SYMBOL\_CLEAR      Deletes all symbols (after confirmation).

SC start      Deletes the symbol 'start'.

## SYMBOL\_LOAD

**Abbreviation:** SL

### Description:

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

### Syntax:

sl <filename>

Parameter	Type	Description
<filename>	String	File name

**Examples:**

SYMBOL\_LOAD TEST.SYM                      Loads the file TEST.SYM.

SL MY\_CODE.SYM                                Loads the file MY\_CODE.SYM.

**SYMBOL\_SAVE****Abbreviation: SS****Description:**

Saves symbols to a file in XLINK Pentica-b format. The symbol file extension default is **.SYM**. If the file name already exists, then a prompt to overwrite the file is displayed.

**Syntax:**

ss <filename>

Parameter	Type	Description
<filename>	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.

**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>	String	Displays the symbols including the specified string pattern

**Examples:**

SYMBOL\_VIEW BUFFER      Displays all symbols containing the word BUFFER.

SV      Displays all the symbols.

**TRACE****Abbreviation: TR****Description:**

Displays the trace buffer contents. The last (most recently executed) cycle in the buffer is 0, and older cycles have negative values.

**Syntax:**

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

Parameter	Type	Description
<start rec>	Numeric	Offset (optional, default = most recent cycle - 9)
<count>	Numeric	Count (optional, default = 10)

**Example:**

TR 0 5      Displays five lines of trace buffer contents starting from the top of the buffer.

**TRACE\_ACQUISITION****Abbreviation: TA****Description:**

Enables or disables trace information acquisition

**Syntax:**

ta <mode>

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

**Examples:**

TRACE\_ACQU ISITION E      Trace information acquisition is enabled.

TA D                      Trace information acquisition is disabled.



# 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**

<b>Message</b>	<b>Contents</b>
Break Access	The break access 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 <b>[Stop]</b> button.
Trace Buffer Full	Since the <b>Break</b> mode was selected by <b>Trace buffer full handling</b> in the <b>Trace Acquisition</b> 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**

<b>Message</b>	<b>Contents</b>
Address Error	One of the following states occurred: <ul style="list-style-type: none"><li>• A PC value was an odd number.</li><li>• An instruction was read from the internal I/O area.</li><li>• Word data was accessed to an address other than a multiple of 2.</li><li>• Longword data was accessed to an address other than a multiple of 4.</li></ul> 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.
Illegal Instruction	Either of the following states occurred: <ul style="list-style-type: none"><li>• A code other than an instruction was executed.</li><li>• MOV.B Rn, @-SP or MOV.B @SP+, Rn was executed.</li></ul> Correct the user program to prevent the error from occurring.
Illegal Operation	Either of the following states occurred: <ul style="list-style-type: none"><li>• In the DAA or DAS instruction, relationship between the C and H flags of CCR, and their relation to the value before compensation were incorrect.</li><li>• Zero-division or overflow was caused by the DIVXU or DIVXS instruction.</li></ul> Correct the user program to prevent the error from occurring.
Memory Access Error	One of the following states occurred: <ul style="list-style-type: none"><li>• A memory area that had not been allocated was accessed.</li><li>• Data was written to a memory area having the write protect attribute.</li><li>• Data was read from a memory area having the read disable attribute.</li><li>• A memory area in which memory does not exist was accessed.</li><li>• Data was written to EEPROM with an instruction other than EEPMOV.</li></ul> 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, and parameter block.



# Section 8 Looking at Your Program

This section describes how to look at your program as source code and assembly language mnemonics. HDI's facilities for dealing with code and symbol information are explained and you will be shown how to look at text files in the user interface.

## 8.1 Compiling for Debugging

In order to be able to debug your program at C/C++ source level, your C/C++ program must be compiled and linked with the debug option enabled.

**Note:** Make sure you have the debug option enabled on your compiler and linker, when you generate an object file for debugging.

If your debug object file does not contain any debugging information, then you can still load it into the debugging platform, but you will only be able to debug at assembly-language level.

## 8.2 Viewing the Code

### 8.2.1 Viewing Source Code

To look at your program's source, choose the [View->Source...] menu option; use the **Ctrl+K** accelerator; or click on the Source Window toolbar button .

Select your source file and click [Open], HDI opens a **Source** window:

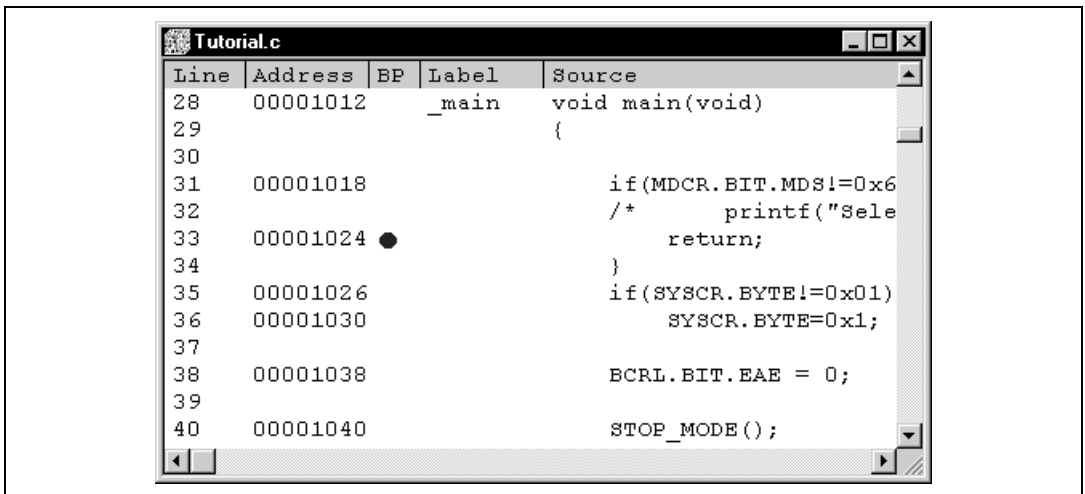




Figure 8.1 Source Window

The **Source** window is divided into two areas; the header bar area and the main window area, and split vertically into five columns; Line, Address, BP (breakpoint), Label, and Source. The respective width of each column can be adjusted by dragging the dividing line between each column title in the header bar. The cursor will change to  and a vertical line will be displayed where the dividing line of the columns will be. Release the mouse button when you are satisfied with the column width and the display will be updated with the new column width.

## 8.2.2 Viewing Assembly-Language Code

If you have a source file open, right-click to open the popup menu and select **Go to Disassembly** to open a **Disassembly** window at the same address as the current **Source** window.

If you do not have a source file, but wish to view code at assembly-language level, either choose the **[View->Disassembly...]** menu option; use the **Ctrl+D** accelerator; or click on the Disassembly Window toolbar button . This will open a **Set Address** dialog box in which you can address to start disassembling.

The **Disassembly** window shows Address, BP (breakpoint), Code - showing the machine code values, Label and Assembler - showing the disassembled mnemonics (with labels when available). Additionally the final column contains any source line starting at that address, thus providing mixed mode display.

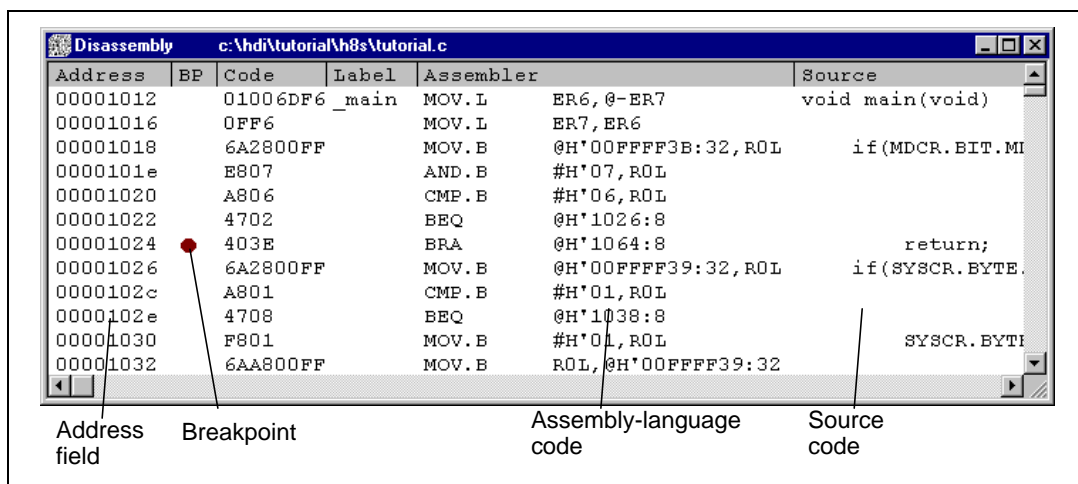
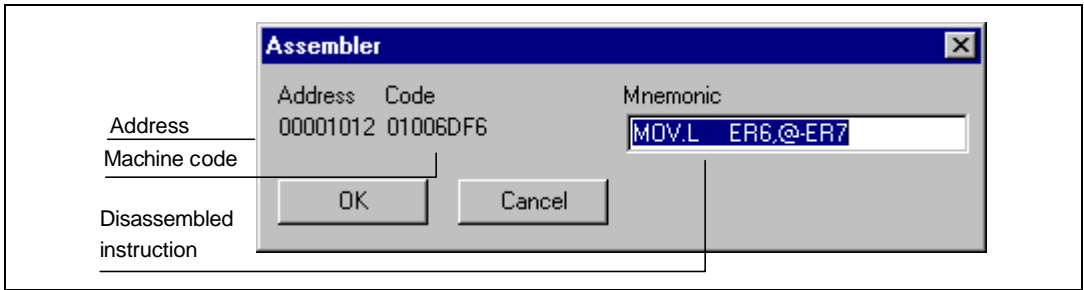


Figure 8.2 Disassembly Window

## 8.2.3 Modifying Assembly-Language Code

You can modify the assembly-language code by double-clicking on the instruction that you wish to change. The **Assembler** dialog box will open:



**Figure 8.3 Assembler Dialog Box**

The address, machine code and disassembled instruction are displayed. Type the new instruction or edit the old instruction in the Mnemonic field. Pressing **ENTER** will assemble the instruction into memory and move on to the next instruction. Clicking [**OK**] will assemble the instruction into memory and close the dialog box. Clicking [**Cancel**] or pressing **ESC** will close the dialog box.

**Note:** The assembly-language display is disassembled from the actual machine code in the debugging platform’s memory. If the memory contents are changed the display will show the corresponding new assembly-language code, but will not match the text shown in the source display.

## 8.3 Looking at Labels

The *debug object file* also contains symbolic information. This is a table of text names that represent an address in the program and is referred to as labels in HDI. You will see symbols in the Label field on the line of the corresponding address, and in the Assembler field as part of an instruction’s operand.

- Notes**
1. An instruction’s operand is replaced with a label name if the operand and label value match. If two or more labels have the same value, then the label that comes first alphabetically will be displayed.
  2. Wherever you can enter an address or value in an HDI edit control you can use a label instead.

### 8.3.1 Listing Labels

To see a list of all the labels defined in the current session open the **Labels** window by choosing the [**View->Labels**] menu option.

BP	Value	Name
	H'00001000	startup_boot
	H'00001012	_main
	H'0000106A	_STOP_MODE
	H'0000108E	_MASK1
	H'000010BA	_DMAC_RUN
	H'00001180	_MASK2
	H'000011AC	_DTC_REGS
	H'0000129E	_DTC_SCI0_ACT
	H'000012E2	_DTC_SCI0_RUN
	H'00001316	_MASK3
	H'00001342	_WDT_RUN
	H'00001358	_DEND0A
	H'0000136C	_WOVI
	H'0000139E	_TXI0
●	H'000013C6	_COPY_MEM
	H'00001412	_INITSCT
	H'0000145C	_D_ROM
	-----	

**Figure 8.4 Labels Window**

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

You can quickly set a software break at an address by double-clicking (or right-clicking and selecting Break on the BP popup menu) in the BP column.

### 8.3.2 Adding a Label from a Source or Disassembly Window

You can quickly add a label from a **Source** or **Disassembly** window, by double-clicking in the Label column at the address for which you want to assign the Label. The **Label** dialog box opens for you to enter the text.



**Figure 8.5 Label Dialog Box**

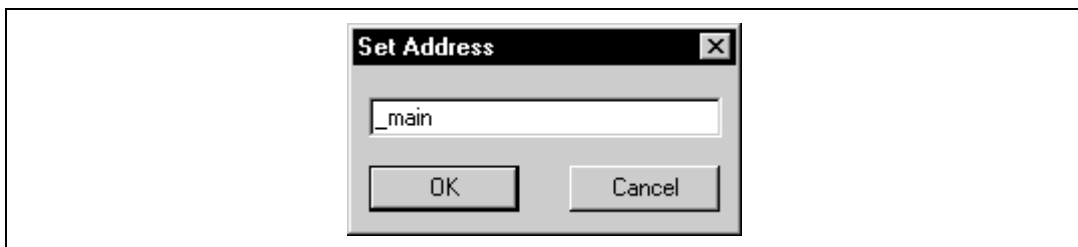
Enter the label name text and click **[OK]**, so that the label is added to the label list with the address value contained in the Address column of the corresponding line, and the **Source** window display is updated to show the label. The **[Clear]** button can be used to remove the label.

This method can also be used for quickly modifying the text of existing labels. When you double-click on the label in the Label column, the text is copied into the edit box of the **Label** dialog box. You can then edit it and the modified version is saved in the label list. The **Source** window display is updated to show the new label.

**Note:** To use added or modified labels again in later sessions, save them in a file. For details, see section 5.6.11, Save As....

## 8.4 Looking at a Specific Address

When you are looking at your program in a **Source** window, you may want to look at another area of your program's code. Rather than scrolling through a lot of code in the program, you can go directly to a specific address. Double-click in the Address column; the **Set Address** dialog box opens:



**Figure 8.6 Set Address Dialog Box**

Enter the address or symbol name in the edit box and either click on [OK] or press **ENTER**. If the code at that address is in the same source file, the **Source** window updates to show the code at the new address. When an overloaded function or a class name is entered, the **Select Function** dialog box opens for you to select a function. For details, refer to section 14, Selecting Functions.

If the new address is in a source file that is already being viewed in a **Source** window, that window is brought to the front and updated to show the code at the new address.

If the new address is in another source file, a new **Source** window opens to show the code at that address. By default the new window shows source if it is available. If no source is available for the new address, then a **Disassembly** window shows assembly-language code.

### 8.4.1 Looking at the Current Program Counter Address

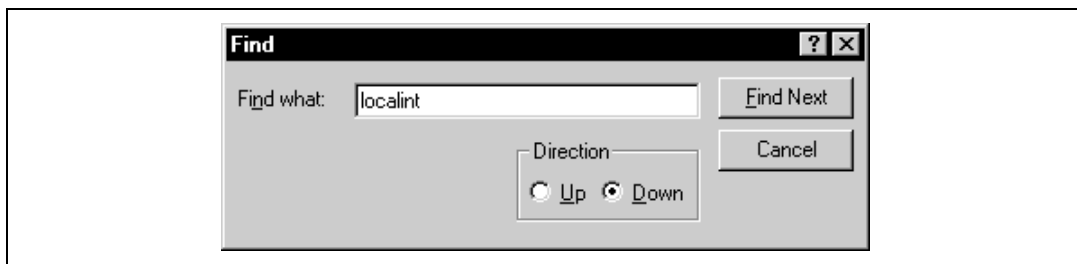
Wherever you can enter an address or value into HDI, you can also enter an expression (see section 2.2, Data Entry). If you enter a register name prefixed by the “#” character, the contents of that register will be used as the value in the expression. Therefore if you open the **Set Address** dialog box and enter the expression “#PC”, the **Source** or **Disassembly** window display will go

to the current PC address. You can also display from an offset of the current PC by entering an expression with the PC register plus an offset, e.g., “#PC+0x100”.

## 8.5 Finding Text

You can search for a particular text string in the **Source** window using the find option. To do this, choose the [**Find...**] menu option from the popup menu, or use the **F3** accelerator key.

The **Find** dialog box is displayed:




**Figure 8.7 Find Dialog Box**

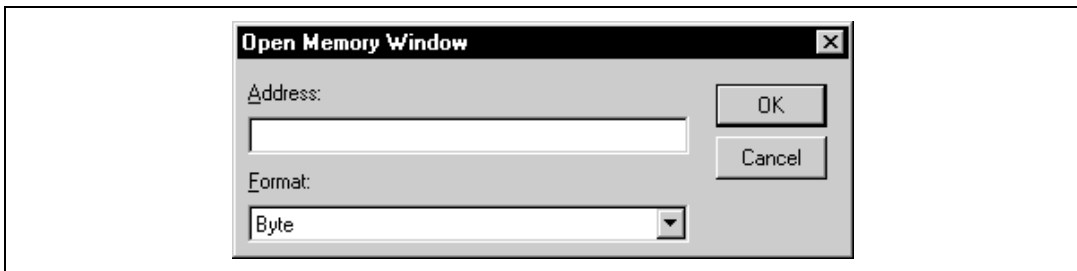
Enter the text that you wish to find and click [**Find Next**] or press **ENTER**. The **Source** window will display the text (if found) highlighted. To find the next occurrence of the text, click [**Find Next**] or press **ENTER** again. To close the **Find** dialog box, click [**Cancel**] or press **ESC**.

# Section 9 Working with Memory

This section describes how to look at areas of memory in the CPU's address space. It will show you how to look at an area of memory in different formats, fill, move and test a block of memory, and save, load and verify an area of memory with a disk file.

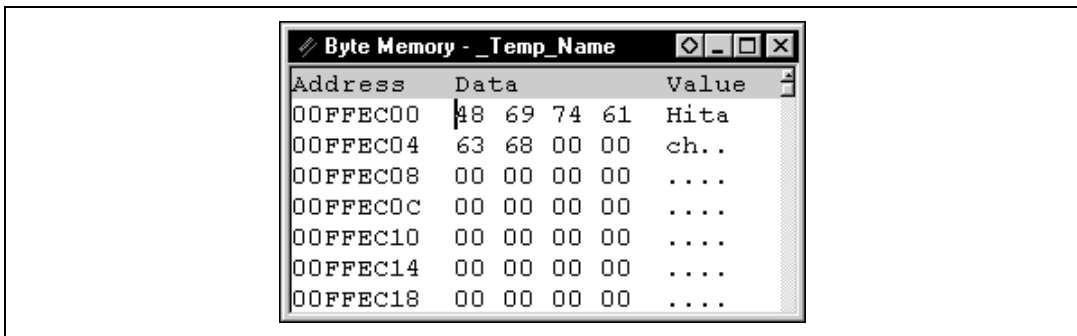
## 9.1 Looking at an Area of Memory

To look at an area of memory, choose the [View->Memory...] menu option; using the **Ctrl+M** accelerator; or clicking the Memory Window toolbar button  to open a **Memory** window. This will open an **Open Memory Window** dialog box:



**Figure 9.1 Open Memory Window Dialog Box**

Type in the start address or equivalent symbol for the window display in the Address field and select the required display format from the Format list. Click [OK] or press **ENTER**, and the dialog box closes and a **Memory** window opens:



**Figure 9.2 Memory Window (Bytes)**

There are two display columns excluding the address display column:

1. **Data** - The data read from the debugging platform. Where supported it is read from physical memory at the displayed width. Editing is supported.

2. Value - Data displayed in an alternative format. Editing is not supported.

If you want to change the display format from the one you selected when you opened the window, do it from the popup menu.

### 9.1.1 Displaying Memory as ASCII

To display and edit memory as ASCII characters, choose the **[ASCII]** menu option from the popup menu and the display will be updated to show the area of memory as ASCII characters.

### 9.1.2 Displaying Memory as Bytes

To display and edit memory as bytes, choose the **[Byte]** menu option from the popup menu and the display will be updated to show the area of memory as individual bytes as shown in figure 9.2.

### 9.1.3 Displaying Memory as Words

To display and edit memory as words, choose the **[Word]** menu option from the popup menu and the display will be updated to show the area of memory as 16-bit words.

### 9.1.4 Displaying Memory as Longwords

To display and edit memory as longwords, choose the **[Long]** menu option from the popup menu and the display will be updated to show the area of memory as 32-bit longwords.

### 9.1.5 Displaying Memory as Single-Precision Floating Point

To display and edit memory as single-precision floating-point data, choose the **[Single float]** menu option from the popup menu and the display will be updated to show the area of memory as single-precision floating-point data.

### 9.1.6 Displaying Memory as Double-Precision Floating Point

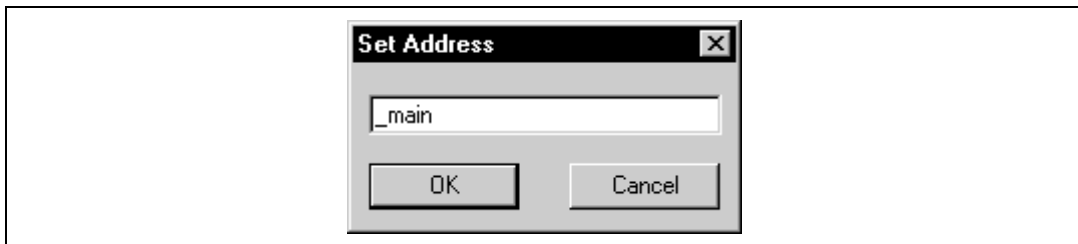
To display and edit memory as double-precision floating-point data, choose the **[Double float]** menu option from the popup menu and the display will be updated to show the area of memory as double-precision floating-point data.

### 9.1.7 Looking at a Different Area of Memory

If you want to change the area of memory that is displayed in the **Memory** window, use the scroll bars. To quickly look at a new address you can use the **Set Address** dialog box. This can



be opened either by choosing the **[Set Address...]** menu option from the popup menu or by double-clicking in the Address column.



**Figure 9.3 Set Address Dialog Box**

Enter the new address value, and click **[OK]** or press **ENTER**. The dialog box closes and the **Memory** window display is updated with the data at the new address. When an overloaded function or a class name is entered, the **Select Function** dialog box opens for you to select a function. For details, refer to section 14, Selecting Functions.

## 9.2 Modifying Memory Contents

There are two ways that you can change the contents of memory at an address:

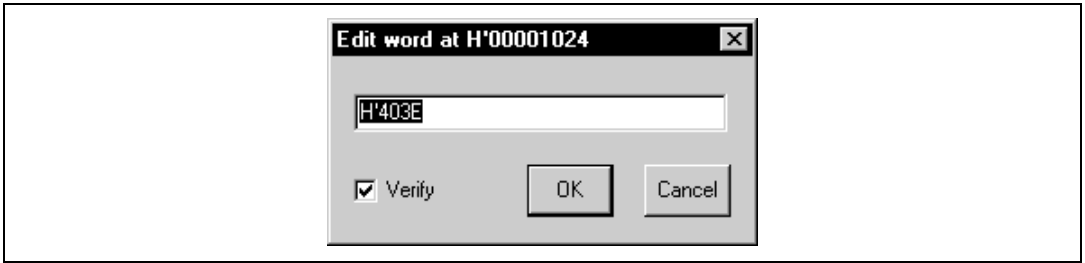
1. Quick edit method - allows you to enter values by typing directly into the window, but is limited to ASCII (when displaying ASCII format) or hexadecimal values only (when displaying all other formats).
2. Full edit method - uses a dialog box to enter values as floating point or evaluated expressions.

### 9.2.1 Quick Edit

The quick way to change the contents of memory is to select the digit that you wish to change, by clicking or dragging on it. You will see the selected digit is highlighted. Type the new value for the digit; it must be in the range 0-9, a-f (when displaying not ASCII format) or the new value for ASCII; it must be ASCII (when displaying ASCII format). The new value is written into the digit and the cursor moves on to the next digit in memory.

### 9.2.2 Full Edit

The full way to change the contents of memory is accessed via the **Edit** dialog box. Move the cursor on the memory unit (depending on your **Memory** window display choice) that you wish to change. Either double-click on the memory unit, or press **ENTER**. The **Edit** dialog box opens:



**Figure 9.4 Edit Dialog Box**

Like any other data entry field in HDI, you can enter a formatted number or C/C++ expression (see section 2.2, Data Entry). When you have entered the new number or expression, click the [OK] button or press ENTER, the dialog box closes and the new value is written into memory.

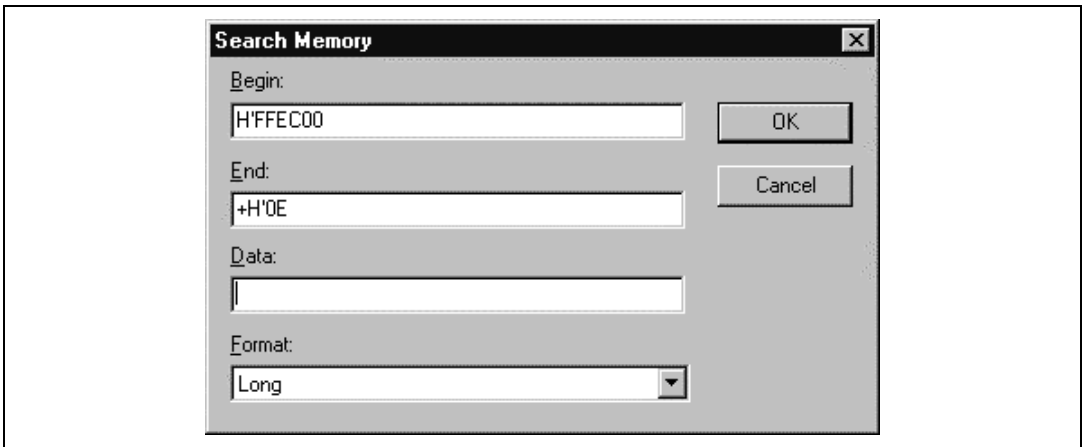
### 9.2.3 Selecting a Memory Range

If the memory address range is in the **Memory** window, you can select the range by clicking on the first memory unit (depending on your **Memory** window display choice) and dragging the mouse to the last unit. The selected range is highlighted.

## 9.3 Finding a Value in Memory

To find a value in memory you must open a **Memory** window, then choose the [Search] menu option from the popup menu. Alternatively, with a **Memory** window in focus, just press F3.

This will open the **Search Memory** dialog box:



**Figure 9.5 Search Memory Dialog Box**

Enter the start and end addresses of the range in which to search (if an area of memory was selected in the **Memory** window then the Begin and End address values will be filled in automatically) and the data value to search for. The end address can also be prefixed by a '+' which will use the entered value as a range.

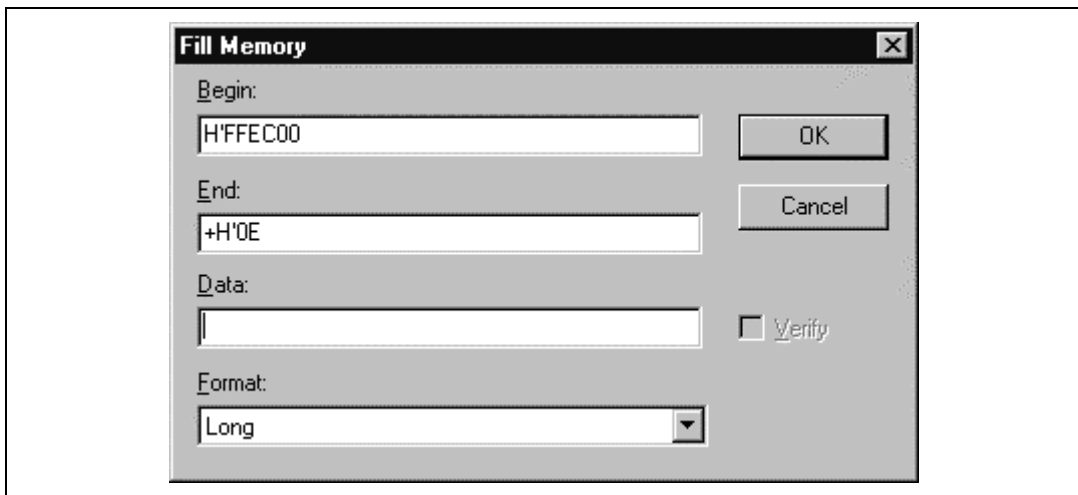
Select the search format and click **[OK]** or press **ENTER**. The dialog box closes and HDI searches the range for the specified data. If the data is found, it will be highlighted in the **Memory** window. If the data cannot be found, the caret position in the **Memory** window remains unchanged and a message informing you that the data could not be found is displayed on the message box.

## 9.4 Filling an Area of Memory with a Value

You can set the contents of a range of memory addresses to a value using the memory fill feature.

### 9.4.1 Filling a Range

To fill a range of memory with the same value, choose the **[Fill...]** menu option on a **Memory** window's popup menu, or **[Memory->Fill...]** menu option. The **Fill Memory** dialog box opens:

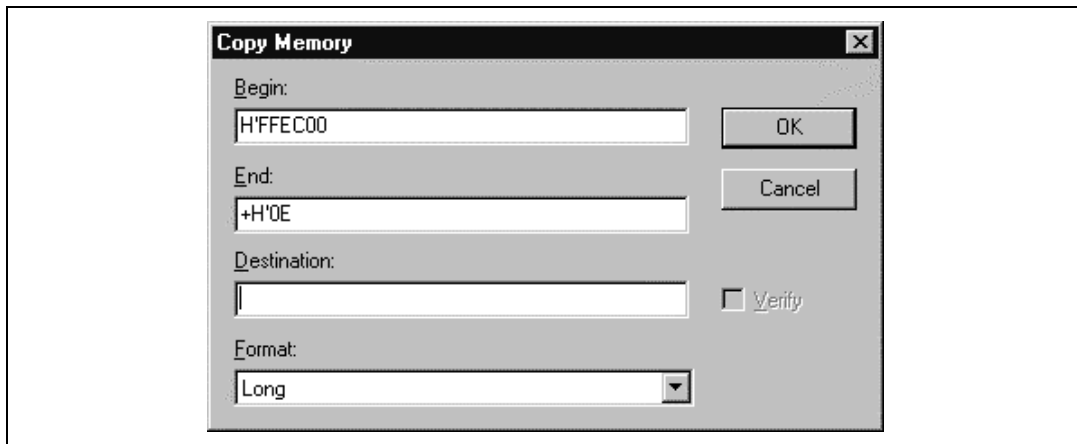


**Figure 9.6 Fill Memory Dialog Box**

If an address range has been selected in the **Memory** window, the specified start and end addresses will be displayed. Select the format from the Format drop list and enter the data value in the Data field. Click the **[OK]** button or press **ENTER**, the dialog box closes and the new value are written into the memory range.

## 9.5 Copying an Area of Memory

You can copy an area of memory using the memory copy feature. Select a memory range (see section 9.2.3, Selecting a Memory Range), choose the **[Copy...]** menu option from the popup menu. The **Copy Memory** dialog box opens:

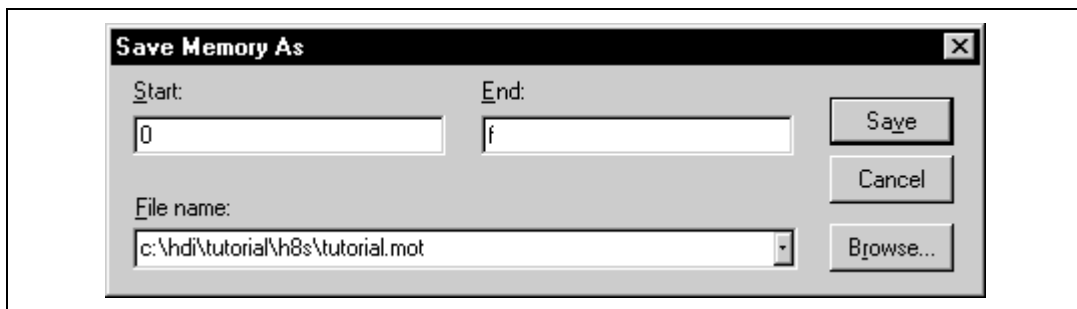


**Figure 9.7 Copy Memory Dialog Box**

The source start and end address specified in the **Memory** window will be displayed in the **Begin** and **End** fields. Enter the destination start address in the **Destination** field and click the **[OK]** button or press **ENTER**, the dialog box closes and the memory block will be copied to the new address.

## 9.6 Saving an Area of Memory

You can save an area of memory in the address space to a disk file using the save memory feature. Open the **Save Memory As** dialog box by choosing the **[Memory->Save...]** menu option:

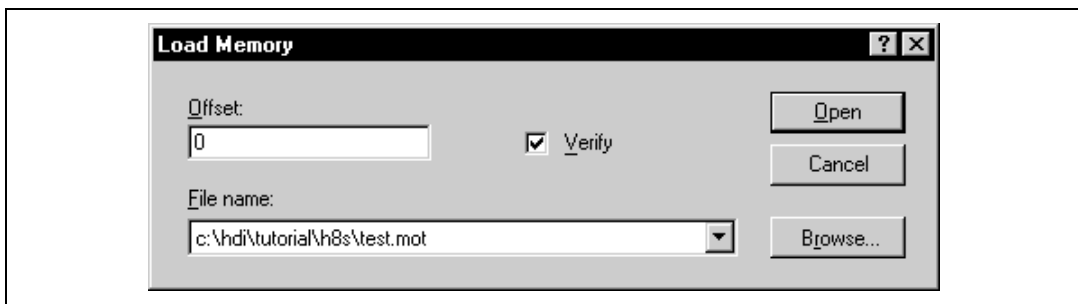


**Figure 9.8 Save Memory As Dialog Box**

Enter the start and end addresses of the memory block that you wish to save and a file name. The File name drop-list contains the previous four file names used for saving memory, or a standard **Save As** dialog box can be launched by clicking the **[Browse...]** button. Click the **[Save]** button or press **ENTER**, so that the dialog box closes and the memory block will be saved to the disk as a Motorola S-Record format file. When the file save is completed, a confirmation message box may be displayed (this can be switched off in the Confirmations tab on the **HDI Options** dialog box).

## 9.7 Loading an Area of Memory

To load an S-Record file to an area of memory without removing the current debugging information by using the load memory feature. Open the **Load Memory** dialog box by choosing the **[Memory->Load...]** menu option:

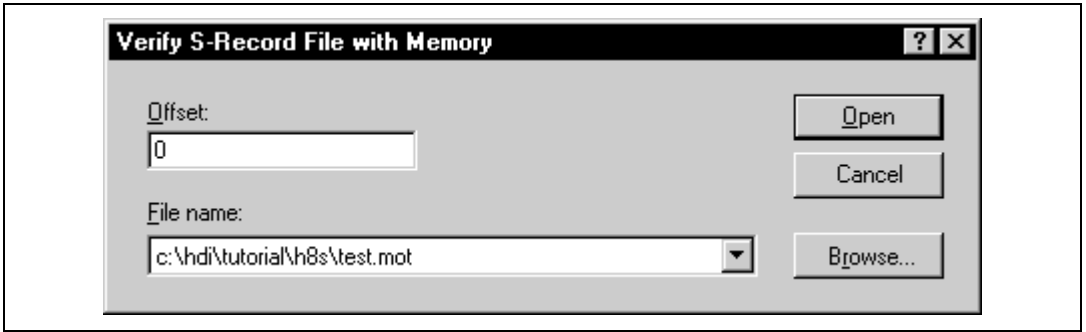


**Figure 9.9 Load Memory Dialog Box**

You can offset the loading address from the address specified in the S-Record by entering a value (positive or negative) in the Offset field. Click the **[Open]** button or press **ENTER**, so that the dialog box closes and the data loads into memory. When the file load is completed, a confirmation message box may be displayed (this can be switched off in the Confirmations tab on the **HDI Options** dialog box).

## 9.8 Verifying an Area of Memory

You can compare an area of memory against a previously saved block of memory using the memory verify feature. Open the **Verify S-Record File with Memory** dialog box by choosing the **[Memory->Verify...]** menu option:




**Figure 9.10 Verify S-Record File with Memory Dialog Box**


You can offset the verification address from the address specified in the S-Record by entering a value (positive or negative) in the Offset field. Click the [**Open**] button or press **ENTER** so that the dialog box closes and the file and the memory contents are verified. When the file verification is completed a confirmation message box may be displayed (this can be switched off in the Confirmations tab on the **HDI Options** dialog box)

# Section 10 Executing Your Program

This section describes how you can execute your program. You can either run your program continuously or step single or multiple instructions at a time.

## 10.1 Running from Reset

To reset your user system and run your program from the reset vector address, choose the [Run->Reset Go] menu option, or click the Reset Go toolbar button [  ].

The program will run until it hits a breakpoint or a break condition is satisfied. You can stop the program manually at any time by choosing the [Run->Halt] menu option, or by clicking the Halt toolbar button [  ].

**Note:** The program will start running from whatever address is stored in the reset vector location. Therefore it is important to make sure that this location contains the address of your startup code.

## 10.2 Continuously Running Your Program

When your program is stopped and the debugger is in break mode, the HDI will highlight the line in the **Source** and **Disassembly** windows that correspond to the CPU's current program counter (PC) address value. This will be the next instruction to be executed if you perform a step or continue running.

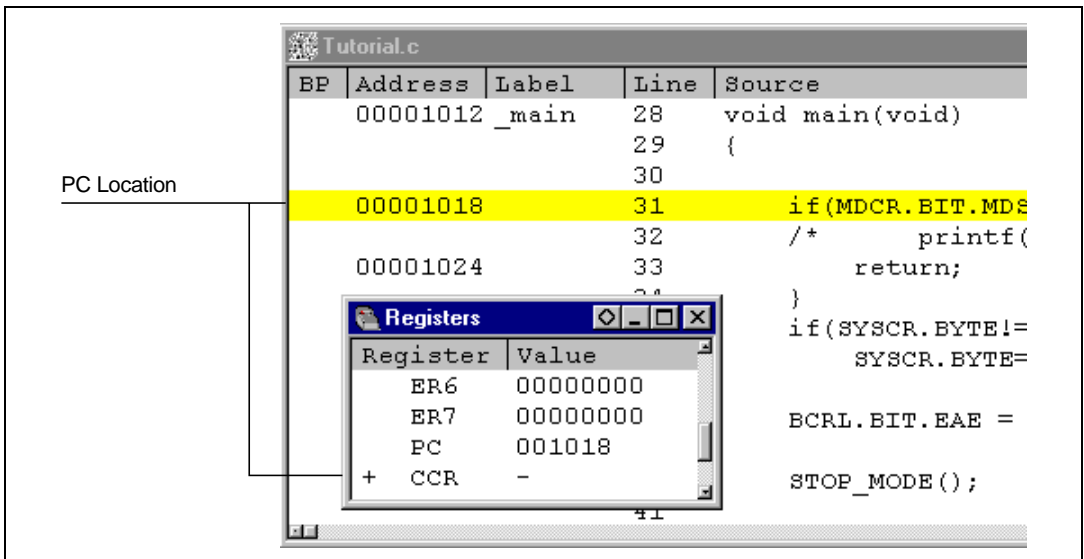



Figure 10.1 Highlighted Line Corresponding to PC Address

To continue running from the current PC address, click the Go toolbar button () or choose the [Run->Go] menu option.


## 10.3 Running to the Cursor

The function for executing only a part of the user program is provided by the Go To Cursor feature to execute to a specific address.

### Using Go To Cursor

1. Make sure that a **Source** or **Disassembly** window is open showing the address at which you wish to stop.
2. Position the text cursor on the address at which you wish to stop by either clicking in the Address field or using the cursor keys.
3. Choose the [**Go To Cursor**] menu option from the popup menu.

The debugging platform will run your program from the current PC value until it reaches the address indicated by the cursor's position.

- Notes**
1. **If your program never executes the code at this address, the program will not stop. If this happens, program execution can be stopped by pressing ESC, choosing the [Run->Halt] menu option, or clicking on the 'Halt' toolbar button ().**
  2. **The Go To Cursor feature requires a temporary breakpoint - if you have already used all those available then the feature will not work, and the menu option will be disabled.**

## 10.4 Running to Several Points


When you want to perform something like the Go To Cursor operation but the destination is outside the **Source** window, or want to stop at several addresses, you can use HDI's temporary breakpoint feature (see section 11.5, Temporary Breakpoints).

## 10.5 Single Step


When you are debugging your code, it is very useful to be able to step a single line or instruction at a time and examine the effect of that instruction on the system. In the **Source** window, a step operation will step a single source line. In the **Disassembly** window, a step operation will step a single assembly-language instruction. If the instruction calls another function or subroutine, you have the option to either step into or step over the function. If the instruction does not perform a call, then either option will cause the debugger to execute the instruction and stop at the next instruction.



### 10.5.1 Stepping Into a Function


If you choose to step into the function, the debugger will execute the call and stop at the first line or instruction of the function. To step into the function either click the Step In toolbar button , or choose the **[Run->Step In]** menu option.

### 10.5.2 Stepping Over a Function Call

If you choose to step over the function, the debugger will execute the call and all of the code in the function (and any function calls that that function may make) and stop at the next line or instruction of the calling function. To step over the function either click the Step Over toolbar button , or choose the **[Run->Step Over]** menu option.

## 10.6 Stepping Out of a Function

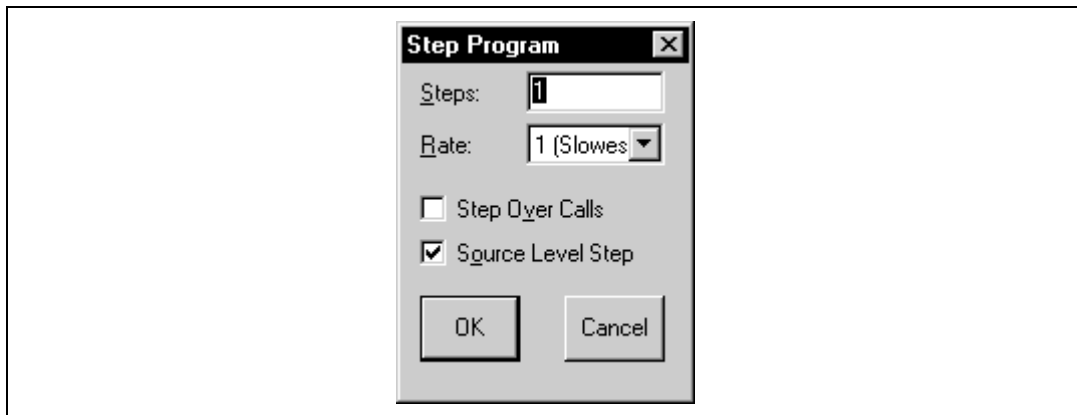
During debugging, there are occasions when you may have entered a function, finished stepping through the instructions that you want to examine, and would like to return to the calling function without tediously stepping through all the remaining code in the function. Or alternatively (and perhaps more usefully) you may have stepped into a function by accident, when you meant to step over it and so want to return to the calling function without stepping all the way through the current function. You can do this with the Step Out feature.

To step out of the current function either click the Step Out toolbar button , or choose the **[Run->Step Out]** menu option.

## 10.7 Multiple Steps

Sometimes you may find it useful to step several instructions at a time. You can do this by using the **Step Program** dialog box. The dialog box also provides an automated step with a selectable intervals between steps. Open it by choosing the **[Run-> Step...]** menu option.

The **Step Program** dialog box is displayed:





**Figure 10.2 Step Program Dialog Box**

Enter the number of steps in the Steps field , choose whether you want to step over function calls by the Step Over Calls check box, and choose whether to make one line of the source program correspond to one step by the Source Level Step check box. If you are using the feature for automated stepping, choose the step rate from the list in the Rate field. Click **[OK]** or press **ENTER** to start stepping.

# Section 11 Stopping Your Program

This section describes how you can halt execution of your program. This section describes how to do this directly by using the halt command and by setting breakpoints at specific locations in your code.

## 11.1 Halting Execution

When your program is running, the Halt toolbar button is enabled  (a red STOP sign), and when the program has stopped it is disabled  (the STOP sign is grayed out). To stop the program click on the Halt toolbar button, press **ESC**, or choose the **[Run->Halt]** menu option.

Your program's execution is halted, with the message "Break = Stop" displayed on the status bar. HDI will then update any open windows.

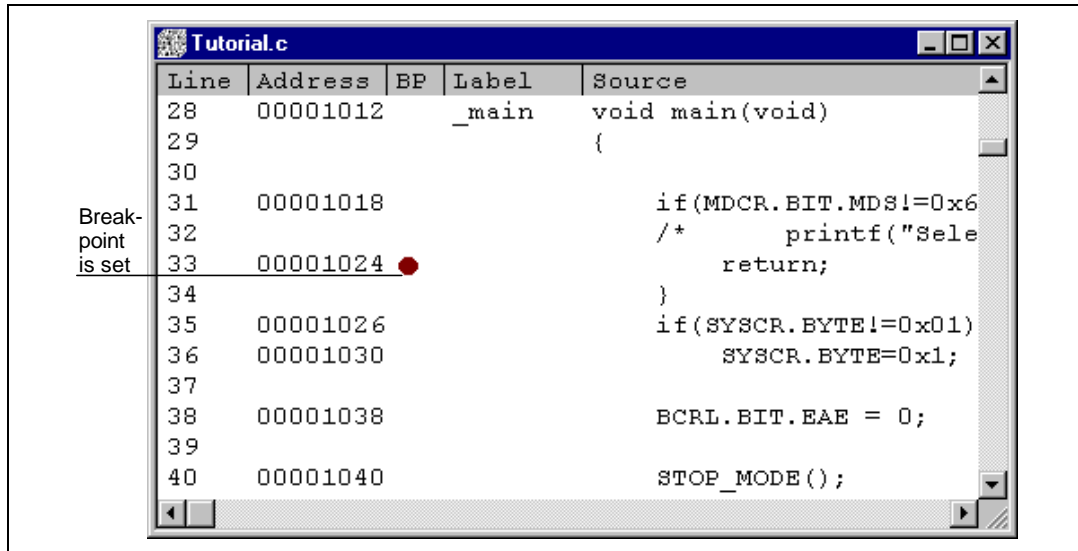
The last break cause can also be viewed in the **Platform** pane of the **System Status** window.

## 11.2 Standard Breakpoints (PC Breakpoints)

When you are trying to debug your program you will want to be able to stop the program running when it reaches a specific point or points in your code. You can do this by setting a PC breakpoint on the line or instruction at which to want the execution to stop. The following instructions will show you how to quickly set and clear simple PC breakpoints. More complex breakpoint operation can be done via the **Breakpoints** window, which is discussed later.

### To set a program (PC) breakpoint

1. Make sure that the Source window is open at the place you want to set a program (PC) breakpoint.
2. Double-click in the BP column, or press F9, at the line showing the address at which you want the program to stop.
3. You will see a circle and the word 'Break' appear in the column to indicate that a program (PC) breakpoint has been set.




**Figure 11.1 Setting a Program Breakpoint**

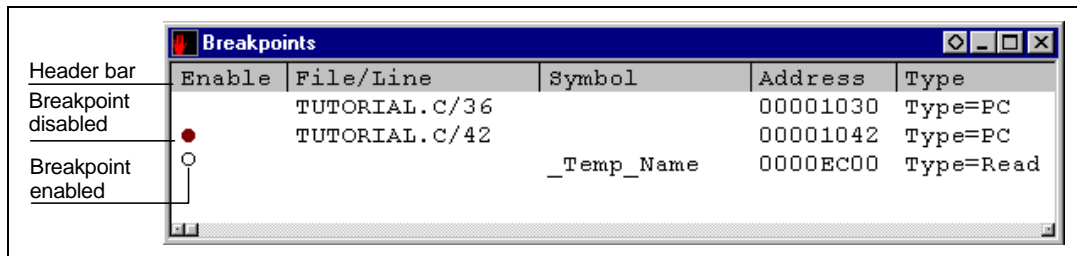
Now when you run your program and it reaches the address at which you set the program (PC) breakpoint, execution halts with the message "Break = PC Breakpoint" displayed on the status bar, and the **Source** window display is updated with the program (PC) breakpoint line highlighted.

**Note:** The line or instruction at which you set a program (PC) breakpoint is not actually executed; the program stops just before it is about to execute it. If you choose to Go or Step after stopping at the program (PC) breakpoint, then the highlighted line will be the next instruction to be executed.

### 11.3 The Breakpoints Window

The **Breakpoints** window allows you to access complex breakpoints (if your debugging platform supports them) and gives you more control over setting or clearing and enabling or disabling breakpoints. To open the **Breakpoints** window choose the [View->Breakpoints] menu option or click the Breakpoint Window toolbar button , if visible.

The **Breakpoints** window opens.



**Figure 11.2 Breakpoints Window**

The window displays a list of the breakpoints set in the system. The breakpoint list is divided horizontally into five columns; Enable, File/Line, Symbol, Address, and Type. The respective widths of each of the columns can be adjusted by clicking and dragging on the dividing line between each column title in the header bar. The cursor will change to  $\leftrightarrow$  and a vertical line will be displayed at the dividing line of the columns. Release the mouse button when you are satisfied with the column width and the display will be updated with the new column width.

### 11.3.1 Adding a Breakpoint

You can add a new breakpoint in the **Breakpoints** window by choosing the **[Add...]** menu option from the popup menu.

The **Set Break** dialog box will open in which you can enter the type and parameters of the new breakpoint.

### 11.3.2 Modifying a Breakpoint

To edit an existing breakpoint in the **Breakpoints** window, select the breakpoint in the list by double-clicking, or by clicking on the line corresponding to it and choose **[Edit...]** menu option from the popup menu.

The **Set Break** dialog box will open in which you can change the type and parameters of the selected breakpoint. When a break sequence is selected, the **Break Sequence** dialog box will open.

### 11.3.3 Deleting a Breakpoint

To delete an existing breakpoint in the **Breakpoints** window, select the breakpoint in the list by clicking on the line corresponding to it and choose the **[Delete]** menu option from the popup menu.

The breakpoint is deleted and the window is updated.

### 11.3.4 Deleting All Breakpoints

To delete all of the breakpoints listed in the **Breakpoints** window choose the **[Delete All]** menu option from the popup menu.

All breakpoints are deleted and the window is cleared.

## 11.4 Disabling Breakpoints

During the course of a debugging session you may find that you tend to focus on particular areas of code for a period of time and then look at other areas, but want to return to the previous ones afterwards. When concentrating on these areas you will want to set breakpoints to stop your program execution at useful points. If you have set these breakpoints and wish to move on to another area of investigation, but know that you will want to return to the current area later, it is frustrating to have to delete all the breakpoints you have set only to have to set them all again when you return. Fortunately, HDI eases this problem by allowing you to disable breakpoints, while still leaving them in the breakpoint list.

### 11.4.1 Disabling a Breakpoint

To disable an individual breakpoint, select the breakpoint in the list by clicking on the line corresponding to it and choose the **[Disable]** menu option from the popup menu.

Alternatively, double-click in the Enable column of the breakpoint you need to disable.

The symbol in the Enable column is cleared to show that the breakpoint is disabled.

### 11.4.2 Enabling a Breakpoint

When you want to re-enable a breakpoint in the **Breakpoints** window list, select the breakpoint in the list by clicking on the line corresponding to it and choose the **[Enable]** menu option from the popup menu.

Alternatively, double-click in the Enable column of the breakpoint you need to enable.

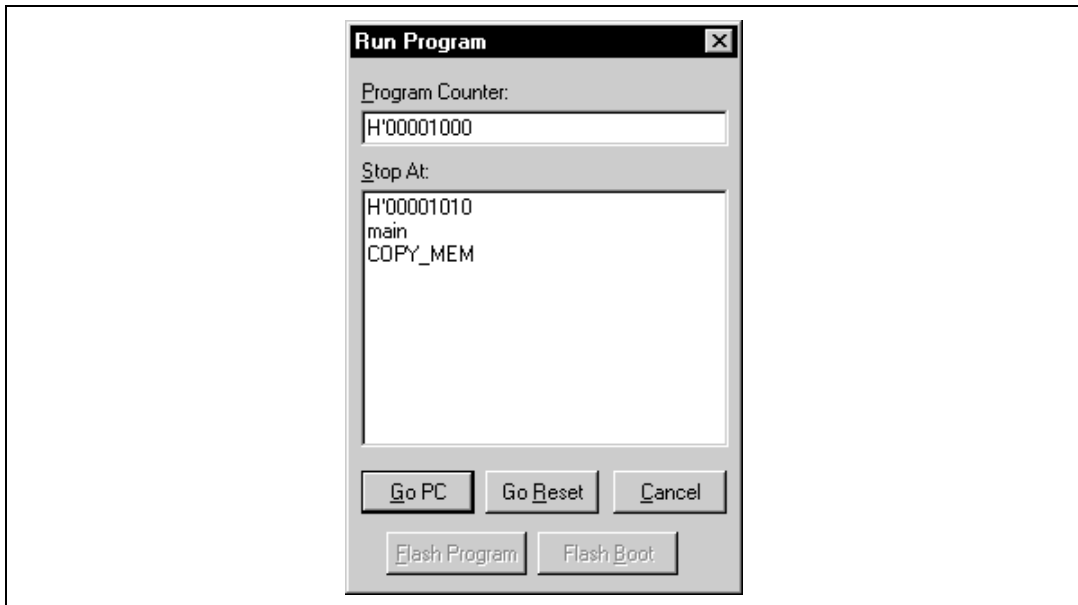
The symbol in the Enable column is set to show that the breakpoint is enabled.

## 11.5 Temporary Breakpoints

There are times when you may want to start running your program and want it to stop if it hits one or more addresses, but do not want to set permanent breakpoints at these addresses. For

example you may want to perform something like the Go To Cursor operation, but the destination may be outside the **Source** window or you may want to stop at several addresses. To do this you can use HDI's temporary breakpoint feature to run as it supports up to ten temporary breakpoints that are cleared when you break. Temporary breakpoints are set in the **Run Program** dialog box, which is opened by choosing the [**Run-> Run...**] menu option.

The **Run Program** dialog box opens:



**Figure 11.3 Run Program Dialog Box**

Enter the symbols or address values for the points at which you want the program to stop (up to ten points) in the Stop At field. When an overloaded function or a class name is entered, the **Select Function** dialog box opens for you to select a function. For details, refer to section 14, Selecting Functions.

Click the [**Go PC**] button to start running from the current program counter address, as displayed in the Program Counter field. Click the [**Go Reset**] button to reset the CPU and start running from the reset vector address.

When the program halts the temporary breakpoints that you specified are cleared from the current breakpoint list. However, when the dialog box is opened again, the list is retained in the Stop At field and will be set again if you click the [**Go PC**] or [**Go Reset**] buttons.





## Section 12 Looking at Variables

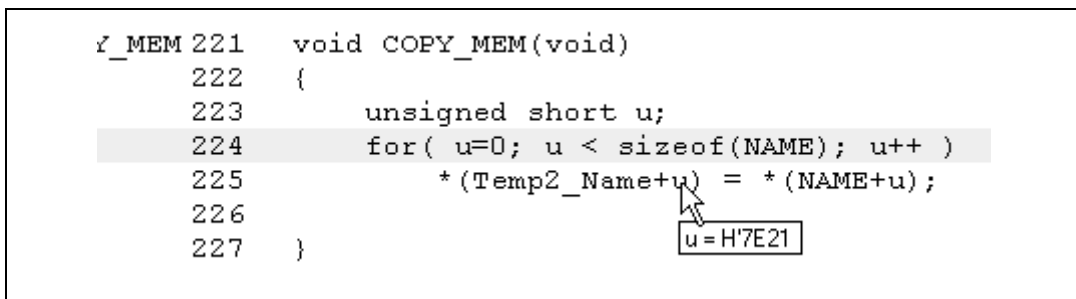
This section describes how to look at the variables and data objects that your program uses. It shows you how to view variables, set up watch items and look at the contents of the CPU's general, FPU, DSP and on-chip peripheral registers.

### 12.1 Tooltip Watch

The quickest way to look at a variable in your program is to use the Tooltip Watch feature.

☛ To use Tooltip Watch:

1. Open the **Source** window showing the variable that you want to examine.
2. Rest the mouse cursor over the variable name that you want to examine; a tooltip will appear near the variable containing basic watch information for that variable.



```

r_MEM 221 void COPY_MEM(void)
      222 {
      223     unsigned short u;
      224     for( u=0; u < sizeof(NAME); u++ )
      225         *(Temp2_Name+u) = *(NAME+u);
      226
      227 }

```

u = H'7E21

Figure 12.1 Tooltip Watch

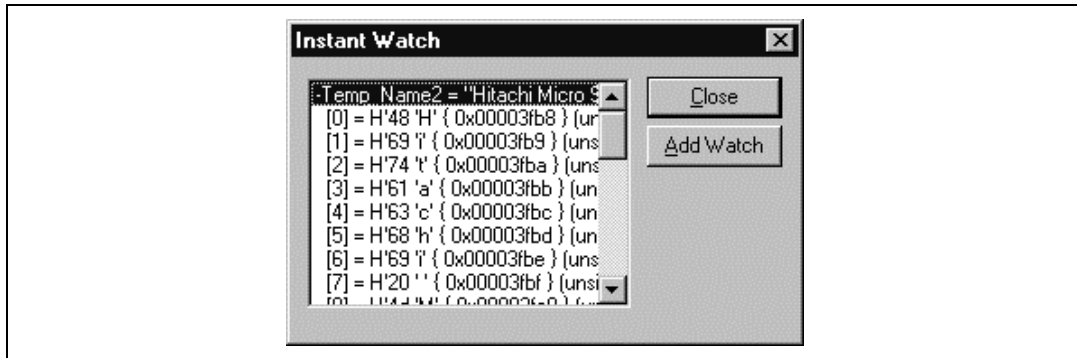
### 12.2 Instant Watch

To look at the variable in more detail, use the Instant Watch feature.

☛ To use Instant Watch:

1. Open the **Source** window showing the variable that you want to examine.
2. Click on the variable. You should see a cursor on the variable.
3. Choose the [**Instant Watch**] menu option from the popup menu.


The **Instant Watch** dialog box opens:



**Figure 12.2 Instant Watch Dialog Box**

You can add this variable to the list of watch items in the **Watch** window by clicking on the **[Add Watch]** button.

## 12.3 Using Watch Items

When you are debugging your program you may find it useful to be able to look at variables of interest and see their values at different times during the program execution. HDI allows you to open **Watch** windows, which contain a list of variables and their values. To open a **Watch** window choose the **[View->Watch]** menu option; or click on the Watch Window toolbar button  if it is visible. A **Watch** window opens. Initially the contents of the window will be blank.

### 12.3.1 Adding a Watch

There are two ways to add watch items to the **Watch** window; the quick method accessed from the **Source** window, and the full method using the **Add Watch** dialog box in the **Watch** window.

#### Quick Method

The quickest way to add a variable to the **Watch** window is to use the Add Watch feature.

- ➔ To use Add Watch from a Source Window:
  1. Open the **Source** window showing the variable that you want to examine.
  2. Click on the variable. You should see a cursor on the variable.
  3. Choose the **[Add Watch...]** menu option from the popup menu.

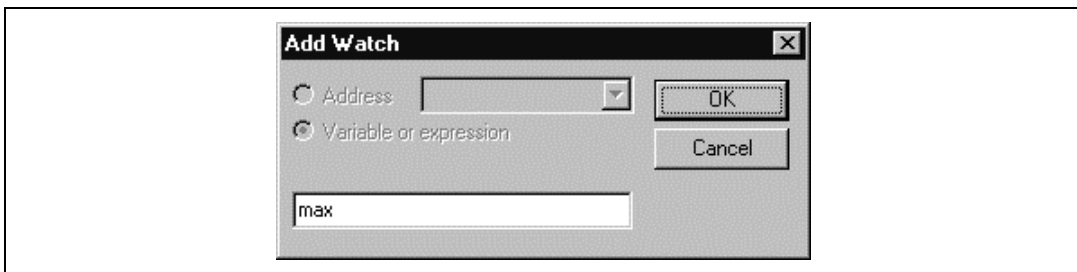
The variable is added as a watch item and the **Watch** window updates.

#### Full Method

The full method uses a dialog box that allows you to enter more complex watch expressions, for example arrays, structures or pointers.

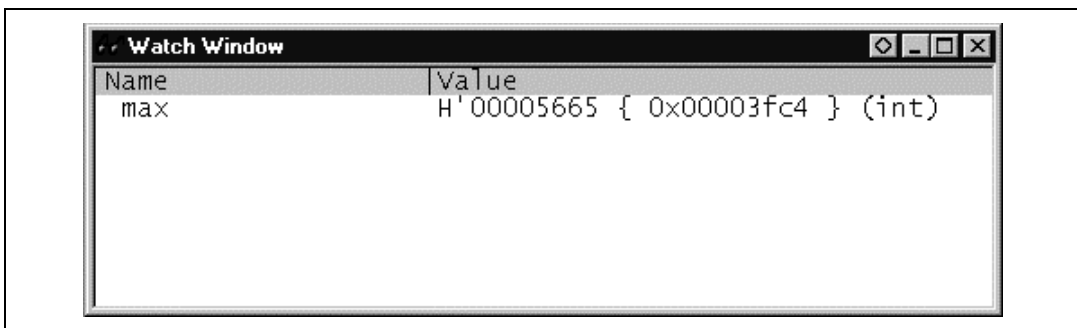
- To use Add Watch from a Watch Window:
  1. Open the **Watch** window.
  2. Choose the **[Add Watch...]** menu option from the popup menu.

The **Add Watch** dialog box opens:



**Figure 12.3 Add Watch Dialog Box**

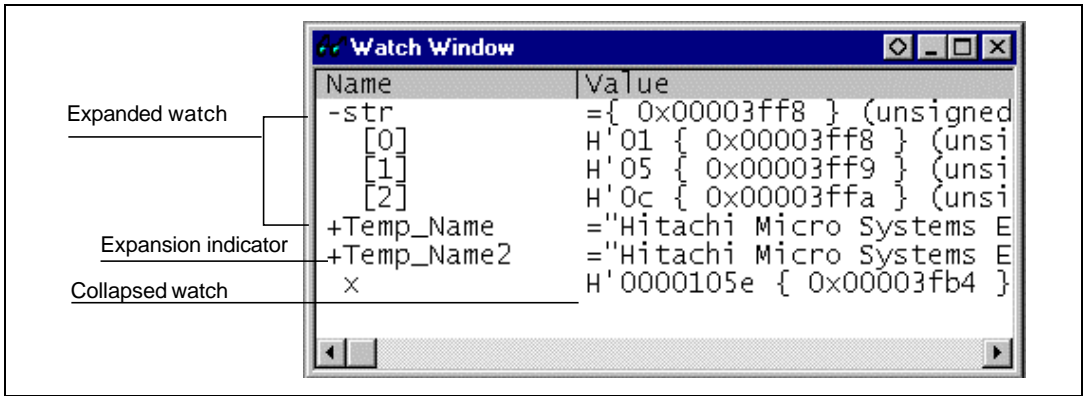
Enter the name of the variable that you wish to watch and click **[OK]**. The variable is added to the **Watch** window.



**Figure 12.4 Watch Window**

### 12.3.2 Expanding a Watch

If a watch item is a pointer, array, or structure, then you will see a plus sign (+) expansion indicator to the left of its name. This means that you can expand the watch item. To expand a watch item, double-click on it. The item expands to show the elements (in the case of structures and arrays) or data value (in the case of pointers) indented by one tab character, and the plus sign changes to a minus sign (-). If the elements of the watch item also contain pointers, structures, or arrays, then they will also have expansion indicators next to them.



**Figure 12.5 Expanding a Watch**

To collapse an expanded watch item, double-click on the item again. The item's elements will collapse back to the single item and the minus sign changes back to a plus sign.

### 12.3.3 Modifying Radix for Watch Item Display

To change the radix of watch item, select the corresponding item by clicking it, and click the right mouse button on the item. Then a popup menu will be displayed. Choose the **[Radix]** menu option from the popup menu. Then choose the radix in which you wish the selected watch item to be displayed. The value will be updated immediately.

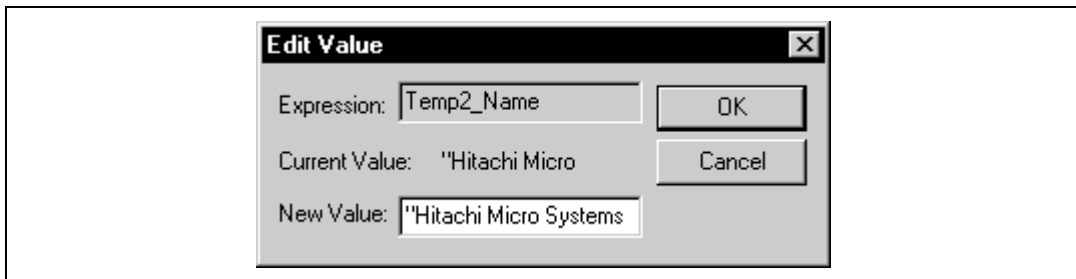
### 12.3.4 Changing a Watch Item's Value

You may wish to change the value of a watch variable, e.g. for testing purposes or if the value is incorrect due to a bug in your program. To change a watch item's value use the Edit Value function.

#### ☛ Editing a watch item's value:

1. Select the item to edit by clicking on it, you will see a blinking cursor on the item.
2. Choose the **[Edit Value]** menu option from the popup menu.

The **Edit Value** dialog box opens:



**Figure 12.6 Edit Value Dialog Box**

Enter the new value or expression in the New Value field and click **[OK]**. The **Watch** window is updated to show the new value.

### 12.3.5 Deleting a Watch

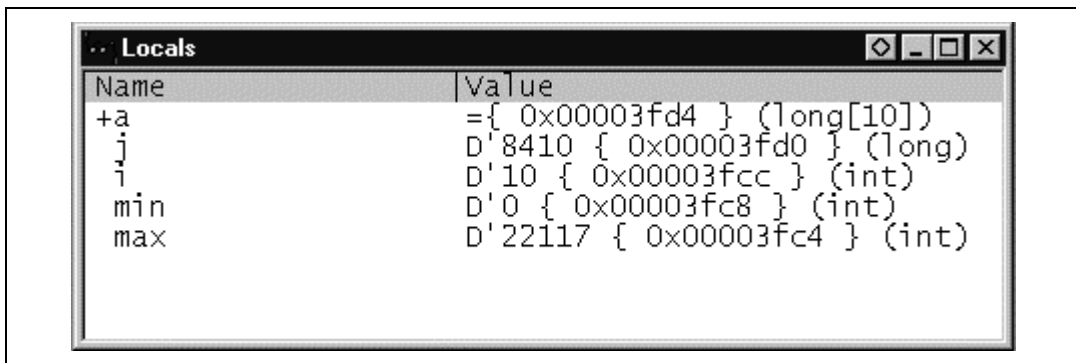
To delete a watch item, select it and choose the **[Delete]** menu option from the popup menu. The item is deleted and the **Watch** window updated.

**Note:** Watch items that you have set in the Watch window can be saved in a session file. See section 15, Configuring the User Interface.

## 12.4 Looking at Local Variables

To look at local variables, open the **Locals** window by choosing the **[View->Locals]** menu option.

The **Locals** window opens:



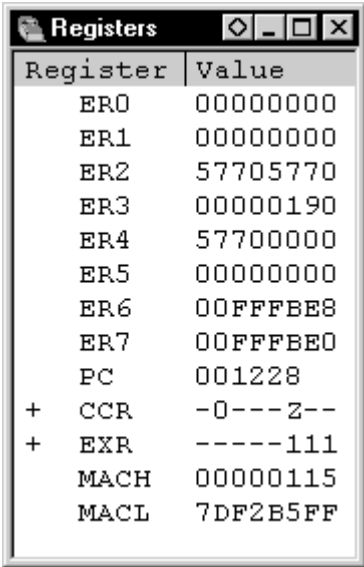
**Figure 12.7 Locals Window**

As you debug your program the **Locals** window will be updated, following a step or break from run, to show the current local variables and their values. If a local variable is not initialized when defined, then the value in the **Locals** window will be undefined until a value is assigned to the local variable.

The local variable values and the radix for local variable display can be modified in the same manner as in the **Watch** window.


## 12.5 Looking at Registers

If you are debugging at assembly-language level, using the **Source** window in assembly language or mixed display, then you will probably find it useful to see the contents of the CPU's general, FPU and DSP registers. You can do this using the **Registers** window.



Register	Value
ER0	00000000
ER1	00000000
ER2	57705770
ER3	00000190
ER4	57700000
ER5	00000000
ER6	00FFFBE8
ER7	00FFFBE0
PC	001228
+ CCR	-0---Z--
+ EXR	-----111
MACH	00000115
MACL	7DF2B5FF

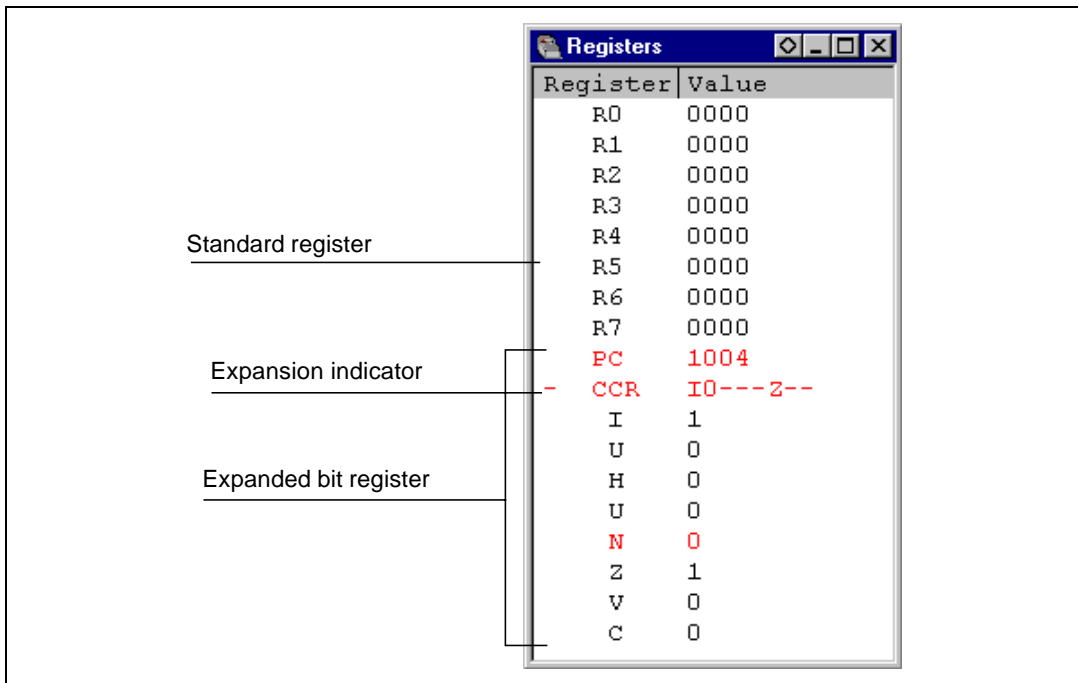
**Figure 12.8 Registers Window**

To open a **Registers** window choose the [View->Registers] menu option or click the CPU Register Window toolbar button . A **Registers** window opens showing all of the CPU's general, FPU and DSP registers and their values, displayed in hexadecimal.

### 12.5.1 Expanding a Bit Register

If a register is used to control or display status using flags at the bit level, then you will see a plus sign (+) expansion indicator to the left of its name; this means that you can expand it. To do this, double-click on the plus sign to show the flags indented by one tab character, and the plus sign

changes to a minus sign (-). If the flags have sub-groups, for example register masks, they will also have expansion indicators next to them.



**Figure 12.9 Expanding a Bit Register**

To collapse an expanded bit register, double-click on the minus sign. The registers collapse back to the single item and the minus sign changes back to a plus sign.

## 12.5.2 Modifying Register Contents

There are two ways that you can change a register's contents. The quick edit method that allows you to enter values by typing directly into the window, but is limited to hexadecimal values only. The full edit method that requires you to enter values via a dialog box, but allows you to enter values in any base and use complex expressions.

### Quick Edit

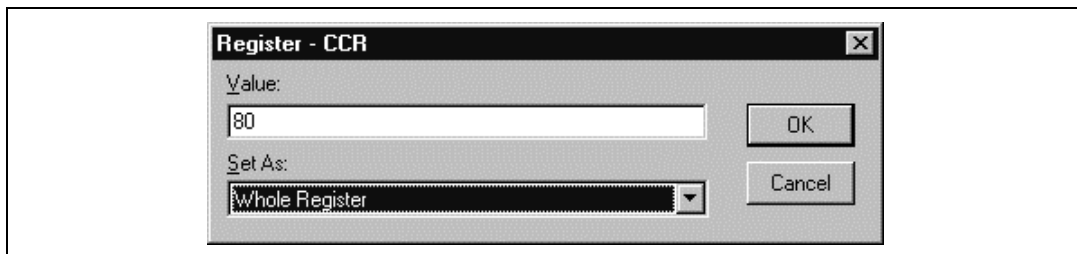
The quick way to change a register's contents is to select the digit that you wish to change, by clicking or dragging on it. You will see the selected digit is highlighted. Type the new value for the digit; it must be in the range 0-9 or a-f. The new value is written into the digit and the cursor moves to the next digit in the register. When you enter a value into the least significant digit of the register, the cursor moves on to the most significant digit of the next register. If the digit of

the register display indicates a bit e.g. in the CPU condition code register (CCR) then you can press **SPACE** to toggle the bit's value.

## Full Edit

The full way to change a register's contents is accessed via a **Register** dialog box. Open the **Register** dialog box in one of three ways:

1. Double-click the register you want to change.
2. Select the register you want to change, and press **ENTER**.
3. Select the register you want to change, and choose the **[Edit...]** menu option from the popup menu.



**Figure 12.10 Register Dialog Box**

As in any other data entry field in HDI, you can enter a formatted number or C/C++ expression (see section 2.2, Data Entry).

You can choose whether to modify the whole register contents (High Word, Low Word, etc), a masked area, floating or flag bits by selecting an option from the drop list box (the contents of this list depend on the CPU model and selected register).

When you have entered the new number or expression, click the **[OK]** button or press **ENTER**. The dialog box closes and the new value is written into the register.

### 12.5.3 Using Register Contents

It can be useful to be able to use the value contained in a CPU register when you are entering a value elsewhere in HDI, for example when displaying a specified address in the **Source** or **Memory** window. You can do this by specifying the register name prefixed by the “#” character, e.g.: #R1, #PC, #R6L, or #ER3.



# Section 13 Overlay Function

Programs making use of the overlay function can be debugged. This section explains the settings for using the overlay function.

## 13.1 Displaying Section Group

When the overlay function is used, that is, when several section groups are assigned to the same address range, the address ranges and section groups are displayed in the **Overlay** dialog box.

Open the **Overlay** dialog box by choosing the [Setup->Overlay] menu option.

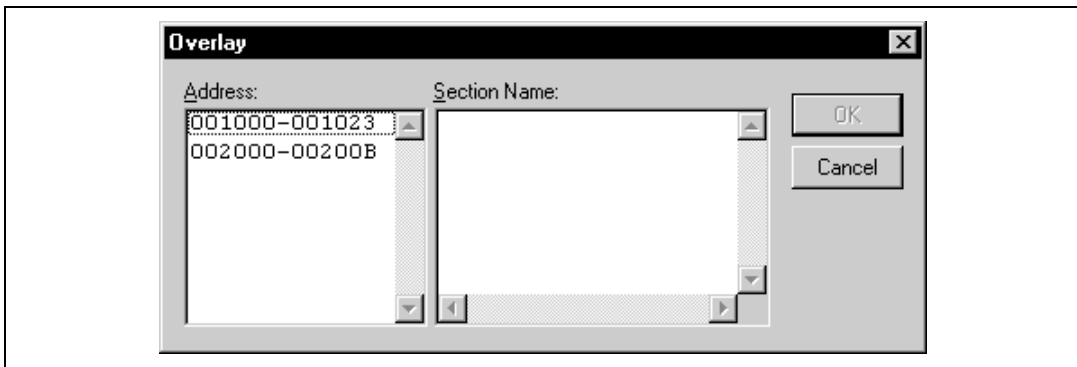


Figure 13.1 Overlay Dialog Box (at Opening)

This dialog box has two areas: the Address list box and the Section Name list box.

The Address list box displays the address ranges used by the overlay function. Click to choose one of the address ranges in the Address list box.

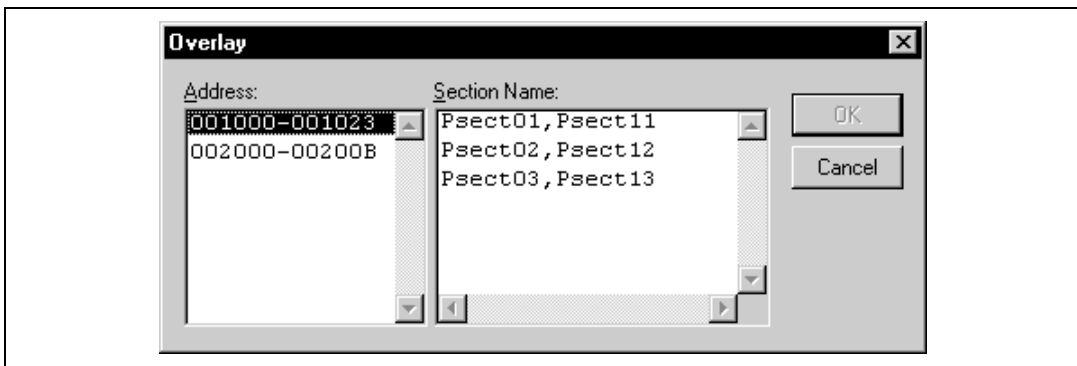


Figure 13.2 Overlay Dialog Box (Address Range Selected)

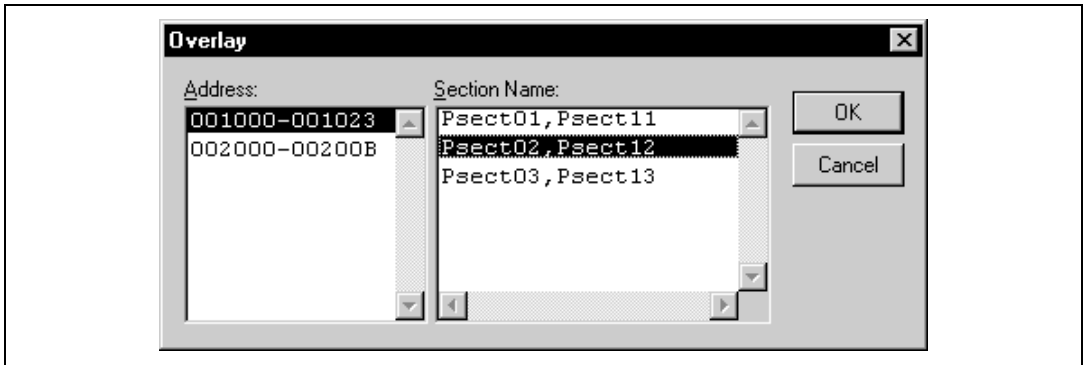
The Section Name list box displays the section groups assigned to the selected address range.

## 13.2 Setting Section Group

When using the overlay function, the highest-priority section group must be selected in the **Overlay** dialog box; otherwise HDI will operate incorrectly.

First click one of the address ranges displayed in the Address list box. The section groups assigned to the selected address range will then be displayed in the Section Name list box.

Click to select the section group with the highest-priority among the displayed section groups.



**Figure 13.3 Overlay Dialog Box (Highest-Priority Section Group Selected)**

After selecting a section group, clicking the [OK] button stores the priority setting and closes the dialog box. Clicking the [Cancel] button closes the dialog box without storing the priority setting.

**Note:** Within the address range used by the overlay function, the debugging information for the section specified in the Overlay dialog box is referred to. Therefore, the same section of the currently loaded user program must be selected in the Overlay dialog box.

# Section 14 Selecting Functions

When selecting overloaded functions or member functions that can be used in C++ programs, follow the description in this section.

## 14.1 Displaying Functions

Use the **Select Function** dialog box to display overloaded functions and member functions.

A function can be selected in the following cases.

- When setting a breakpoint
- When specifying a function in the **Run Program** dialog box
- In the **Set Address** dialog box for opening the **Source** window
- In the **Set Address** dialog box for opening the **Memory** window
- When adding or modifying a symbol
- When specifying a function for performance analysis

When overloaded functions have the same specified function name, or when a class name including a member function is specified, the **Select Function** dialog box opens.

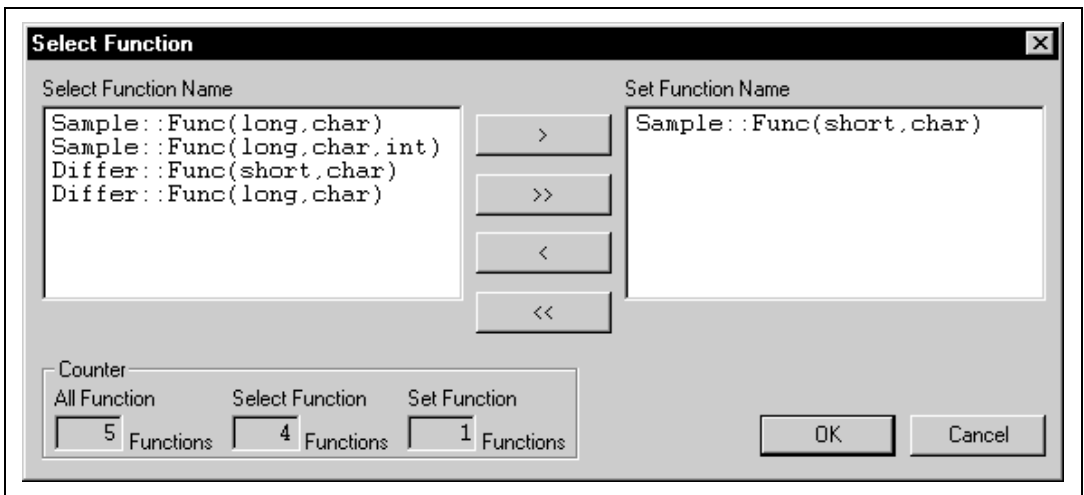


Figure 14.1 Select Function Dialog Box

This dialog box has three areas.

- Select Function Name list box  
Displays the overloaded functions or member functions and their detailed information.
- Set Function Name list box

Displays the function to be set and their detailed information.

- Counter group edit box

All Function      Displays the number of functions with the same name or member functions.

Select Function    Displays the number of functions displayed in the Select Function Name list box.

Set Function        Displays the number of functions displayed in the Set Function Name list box.

## 14.2 Specifying Functions

Select overloaded functions or member functions in the **Select Function** dialog box. Generally, one function can be selected at one time; only for setting breakpoints, setting the function in the **Run Program** dialog box, or setting the function of the performance analysis, more than one function can be selected.

### 14.2.1 Selecting a Function

Click the function you wish to select in the Select Function Name list box, and click the [>] button. You will see the selected function in the Set Function Name list box. To select all functions in the Select Function Name list box, click the [≫] button.

### 14.2.2 Deleting a Function

Click the function you wish to delete from the Set Function Name list box, and click the [<] button. To delete all functions in the Set Function Name list box, click the [≪] button.

### 14.2.3 Setting a Function

Click the [OK] button to set the functions displayed in the Set Function Name list box. The functions are set and the **Select Function** dialog box closes.


Clicking the [Cancel] button closes the dialog box without setting the functions.

# Section 15 Configuring the User Interface

When we designed the user interface for HDI we tried to make all the frequently used operations quickly accessible and have related operations grouped in a logical order. However, when you are in the middle of a heavy debugging session you may find it more useful to have a different arrangement of the user interface items or you may just have a personal preference for the way you want it arranged. We realize this and so HDI allows you to customize the user interface so that you can be satisfied with the tool that you are using for debugging your program. This section describes how you can arrange the user interface windows, customize various aspects of the display and save the configuration.

## 15.1 Arranging Windows

### 15.1.1 Minimizing Windows

If you have temporarily finished using an open window but want to be able to look at it in its current state later, you can reduce it to an icon. This is called *minimizing* the window. To minimize a window, either click on the minimize button of the window, or choose the [  -> **Minimize** ] window menu option.

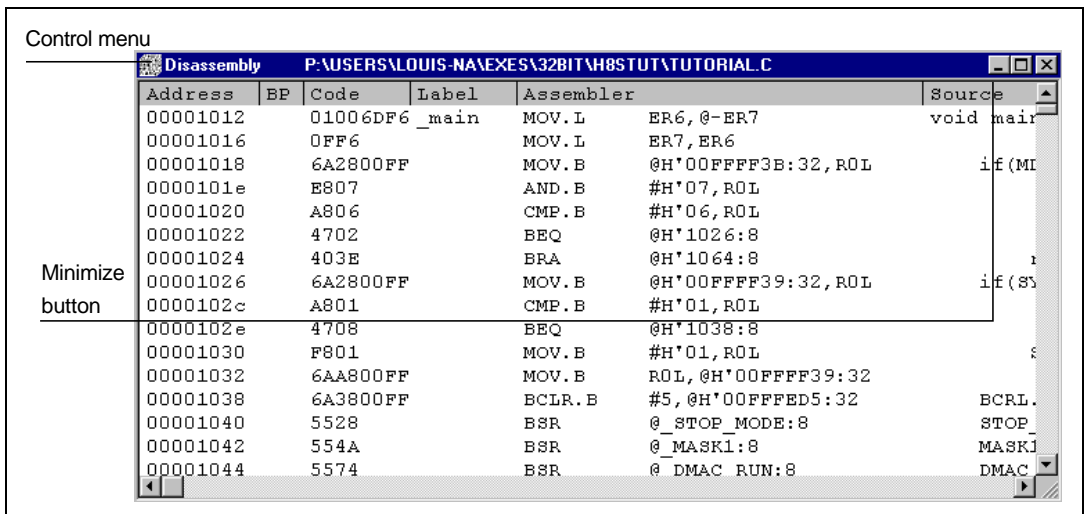


Figure 15.1 Minimizing a Window

The window is minimized to an icon at the bottom left of the HDI application window; for the above **Disassembly** window example, the icon is:



Figure 15.2 Disassembly Window Icon

**Note:** You may not be able to see the icon if you have a window open over the bottom of the screen.

To restore the icon back to a window, either double-click on the icon, or choose the **[Restore]** menu option from the control menu.

### 15.1.2 Arranging Icons

Although the icons will be put at the bottom left of the HDI application window by default when you minimize a window, you can move them anywhere you like in the application window by simply clicking and dragging them to a new position. When you restore the icon to a window, the window will be at the same position that it was in when you minimized it. Similarly, when you minimize it again, the icon will be placed at the last position that you moved it to.

When you have many minimized windows as icons, the display can look rather messy. To tidy up the icons, choose the **[Window->Arrange Icons]** menu option.

The icons will be arranged in order from the bottom left of the application window:

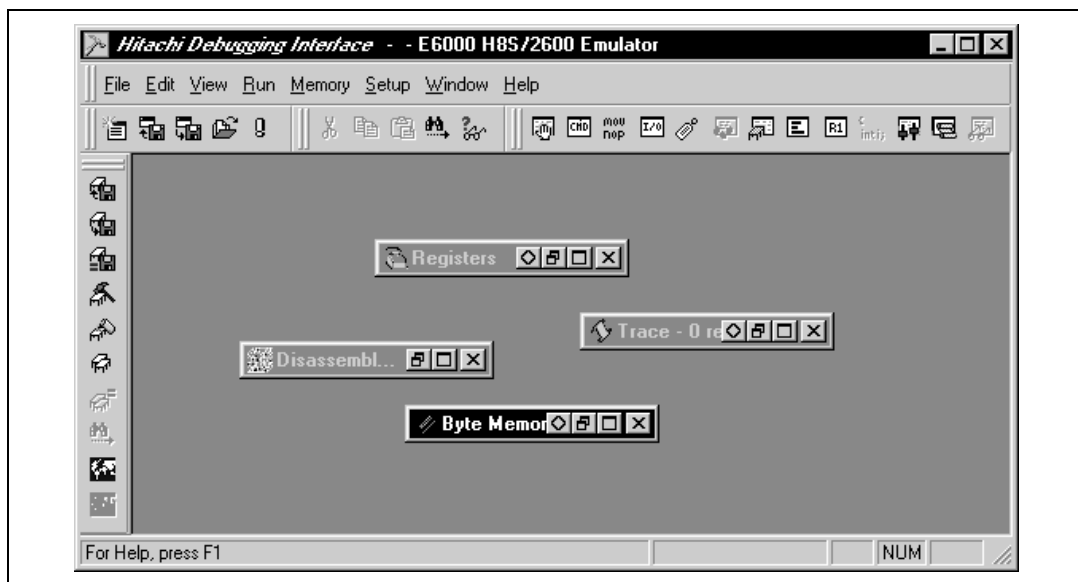


Figure 15.3 Icons Before Arrangement

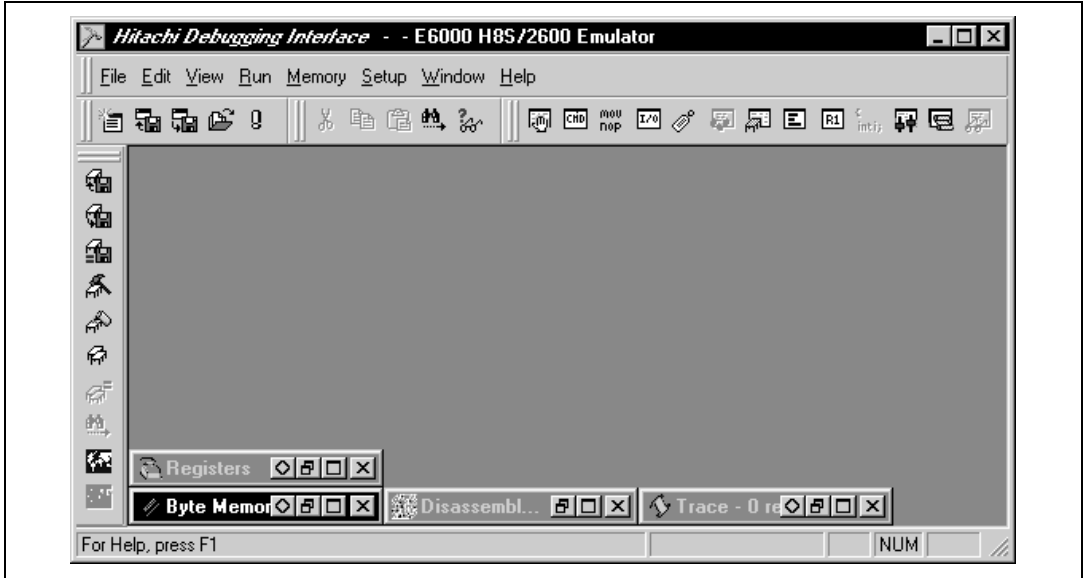


Figure 15.4 Icons After Arrangement

### 15.1.3 Tiling Windows

After some heavy debugging you may find that you have many windows open on the screen. You can arrange all the windows in a tile format with none of them overlapping each other using the Tile function by choosing the **[Window->Tile]** menu option.

All currently open windows are arranged in a tile format. Windows that are minimized to icons are not affected.

### 15.1.4 Cascading Windows

Open windows can also be arranged in a cascading format with only their left and top border visible under the window in front of them by choosing the **[Window->Cascade]** menu option. All currently open windows are arranged in a cascading format. Windows that are minimized to icons are not affected.

## 15.2 Locating Currently Open Windows

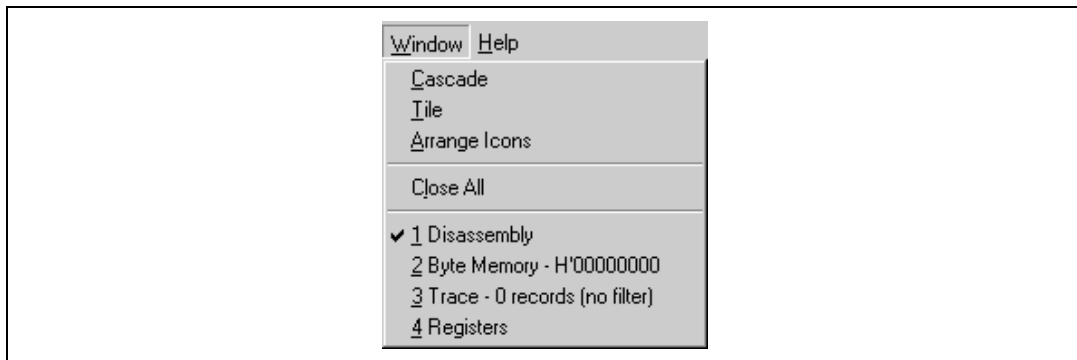
When you have many windows open in the HDI application window it is quite easy to lose one of them behind the others. There are two methods that you can use to find the lost window:

### 15.2.1 Locating the Next Window

To bring the next window in the window list to the front of the display, choose **[Next]** from the window menu, or press **CTRL+F6**. Repeating this operation will cycle selection of all windows (open and minimized).

### 15.2.2 Locating a Specific Window

To select a specific window, choose from the list of windows (open and minimized) at the bottom of the **[Window]** menu. The currently selected window has a check mark next to it in the window list. In the following example, the **Disassembly** window is the currently selected window:



**Figure 15.5** Selecting a Window

The window that you select will be brought to the front of the display. If it is minimized the icon is restored to a window.

## 15.3 Enabling/Disabling the Status Bar

You can select whether or not the status bar is displayed at the bottom of the HDI application window; by default it will be displayed. To disable display of the status bar, choose the **[Setup->Status Bar]** menu option.

The status bar will be disabled and removed from the HDI application window display. To re-enable the Status bar display, choose the **[Setup->Status Bar]** menu option again. The Status bar will be enabled and added to the HDI application window display.



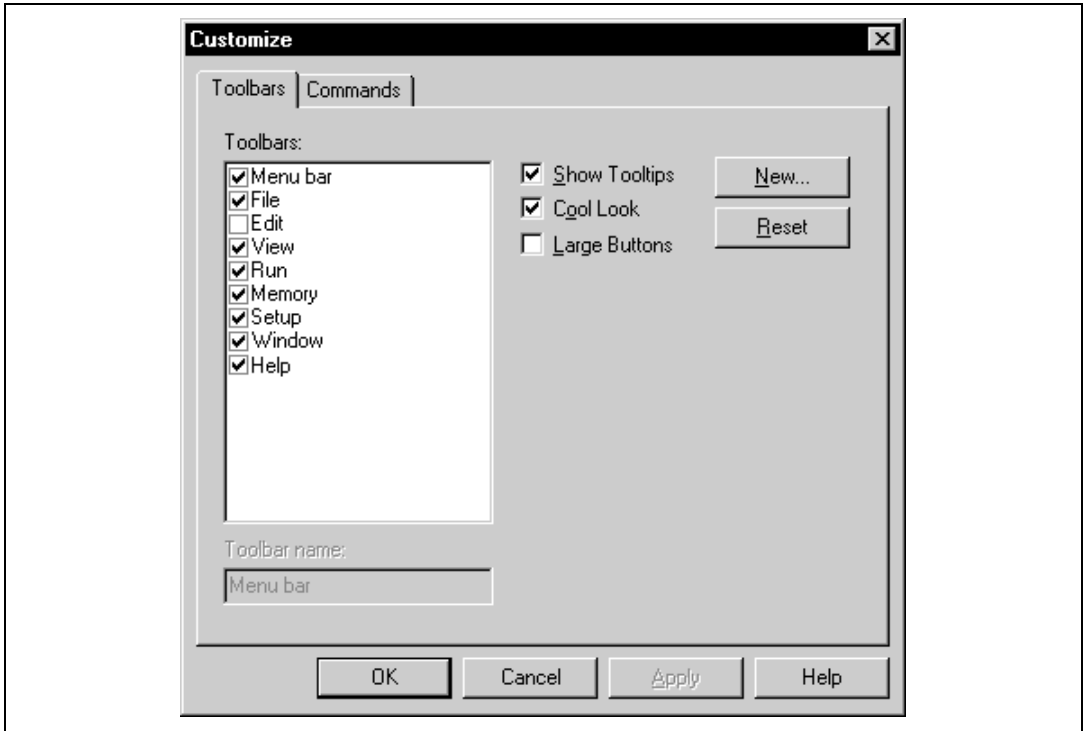
## 15.4 Customizing the Toolbar

To control the selection and arrangement of buttons displayed on the toolbar, choose the [**Setup->Customize->Toolbar...**] menu option.

The **Customize Toolbar** dialog box opens and contains two panes. The first pane 'Toolbars' is used to set the overall appearance of the toolbars, while the second pane 'Commands' is used to set the individual buttons in each toolbar.

### 15.4.1 Overall Appearance

Select the Toolbars pane to set the overall appearance of the toolbars:



**Figure 15.6 Customize Toolbar (Toolbars) Dialog Box**

The toolbars are listed in a multi-selection list box. To individually switch off a toolbar, clear the check box next to the name (this name is displayed in a mini-title bar when the toolbar is not attached to the border of the main frame window).

**Note:** The menu bar cannot be switched off.

If you need to conserve desktop area (for example, when using a portable) then clear the 'Cool Look' check box to revert to the classic Windows® 3.1 style menu and toolbars.

It is possible to add user-defined toolbars - click on the [New...] button and enter a name for your toolbar. This can be edited later in the Toolbar Name edit box (feature only available for user defined toolbars). The new toolbar, in this case called 'My Toolbar', will appear floating at the top-left of the main frame but will have no buttons. To add buttons, you will now have to customize your toolbar.

### 15.4.2 Customizing Individual Toolbars

Customizing individual toolbars requires a mouse or other pointing device. The feature is not available if only the keyboard is available. This is because the toolbars only operate with a mouse, so customizing them would be unnecessary unless you have a mouse.

Select the Commands pane to set the individual buttons in each toolbar:

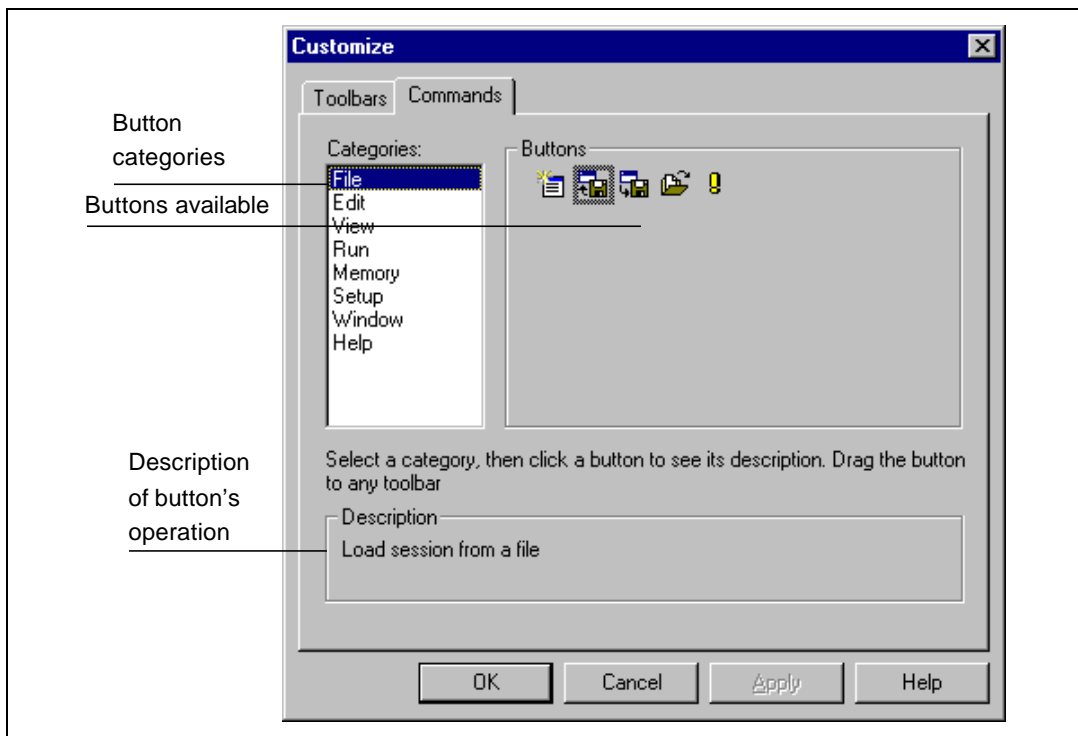


Figure 15.7 Customize Toolbar (Commands) Dialog Box

### 15.4.3 Button Categories

At the top left of the dialog box is a list of button categories. For each category a list of buttons within that category will be displayed to the right. Click on a button operation option in the list to view a description of the button's operation in the Description field.

### 15.4.4 Adding a Button to a Toolbar

➡ To add a button to a toolbar:

1. Select the button category from the button category list.
2. Select the button item from the operation list.
3. Drag the button from the dialog box to the toolbar location you wish to add the new button. Then the button is inserted into the tool bar.

### 15.4.5 Positioning a Button in a Toolbar

➡ To move a button position in a toolbar:

1. Select the button in a toolbar.
2. Drag the button to the new position in the toolbar or another toolbar.

**Note:** Holding down the Ctrl key while dragging will copy the button.

### 15.4.6 Removing a Button from a Toolbar

➡ To remove a button in a toolbar:

1. Select the button in a toolbar.
2. Drag the button out of the toolbar (anywhere into the main frame).

## 15.5 Customizing the Fonts

You can customize the display font for text style windows (e.g. **Source** and **Memory** windows), or change the default font that is used when a new window is opened.

To change the display font, choose the [**Setup->Customize->Font**] menu option. This will launch the **Font** dialog box:

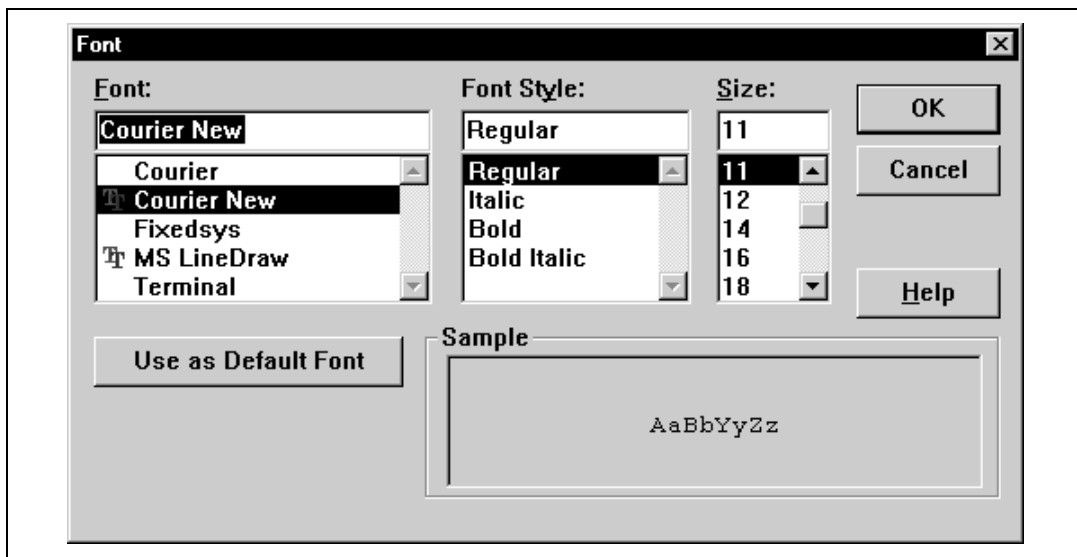


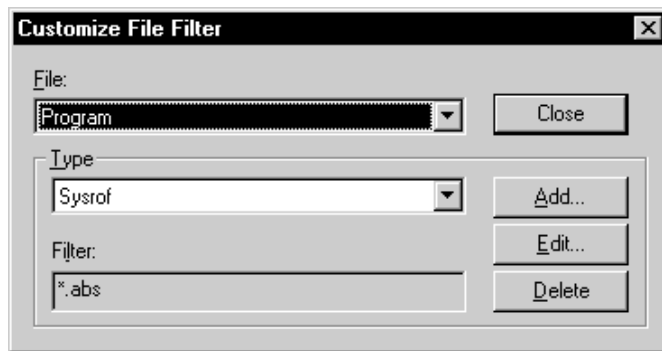
Figure 15.8 Font Dialog Box

The dialog box is based on the standard Windows® font selection dialog box, except that only fixed width fonts are listed in the Font list box. By pressing the [Use as Default Font] button, the font to be used when a new window is opened can be specified.

## 15.6 Customizing the File Filters

You can customize the file filters displayed in the **Open** dialog box.

To change the filters, choose the [**Setup->Customize->File Filter**] menu option. This will launch the **Customize File Filter** dialog box:



**Figure 15.9 Customize File Filter Dialog Box**

**Note:** Changes are made immediately when using this dialog box. There is no option to cancel changes made.

- To edit an existing filter:
  1. Select the file group from the File drop list.
  2. Select the file type name from the Type drop list.
  3. Click the [**Edit...**] button to open the **Edit Filter** dialog box. The dialog title will display the file group that is being changed. The edit box on this dialog box is limited to accept only valid characters for filter type or extension.
  4. Change the filter name and/or extension. If more than one extension is required, then separate each extension with a semi-colon. For example:  
\*.mot;\*.a20;\*.a37
- To enter a new filter:
  1. Select the file group from the File drop list.
  2. Click the [**Add...**] button to open the **Add Filter** dialog box. The dialog title will display the file group that is being changed. The edit box on this dialog box is limited to accept only valid characters for filters.
  3. Enter a name for the filter type and the extensions you want to use for the filter.

**Note:** If the filter type entered matches an existing type, the filter for the existing type will be changed to the newly entered filter.

- To remove a filter:
  1. Select the file group from the File drop list.
  2. Select the file type name from the Type drop list.
  3. The file type will be removed when the [**Delete**] button is clicked.

## 15.7 Saving a Session

If you have downloaded the user program into the debugging platform, have the corresponding source files displayed and a number of auxiliary windows open, then it can take some time to setup this information the next time the program is loaded. To help with this, HDI can save the current settings to a file.

If you are already using a named session, or want to create a session with the same name as the current object file, choose the **[File->Save Session]** menu option.

To save the current setting under a new name, choose the **[File->Save Session As...]** menu option. This will launch a common file dialog box prompting you for a file name. Up to three files are saved; an HDI session file (\*.hds); a target session file (\*.hdt); and a watch session file (\*.hdw). The first includes the HDI interface settings, e.g. all the open windows and their positions. The second includes the settings specific to the debugging platform/user system, e.g. the name of the debugging platform and its configuration. The third is only created if a **Watch** window is open and it includes a list of the variables currently being watched.

The session name is then displayed as the second entry in HDI's title bar.

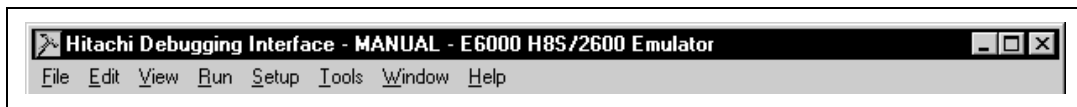


Figure 15.10 Session Name Display

**Note:** The session file does not include symbol or memory information. To use modified information again in later sessions, save the symbol and memory information in appropriate files. For details, see section 9.6, Saving an Area of Memory and section 5.6.11, Save As....

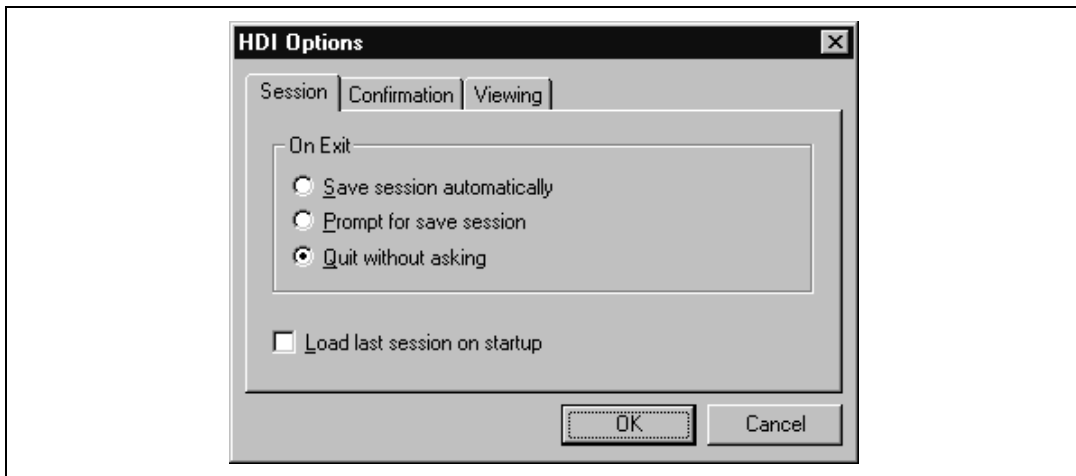
## 15.8 Loading a Session

To reload a saved session, choose the **[File->Load Session...]** menu option. This will launch a standard Windows® file dialog box prompting you for an HDI session file name (\*.hds).

Any currently open windows will be closed, and the connection to the debugging platform initialized. If user program has been downloaded to the user system, then the status bar will display the percentage done. When the download is completed, windows will be opened and updated to show the latest information from the user system.

## 15.9 Setting HDI Options

There are a number of settings available to help you to use the HDI interface. Choosing the [Setup->Options...] menu option will launch the **HDI Options** dialog box:

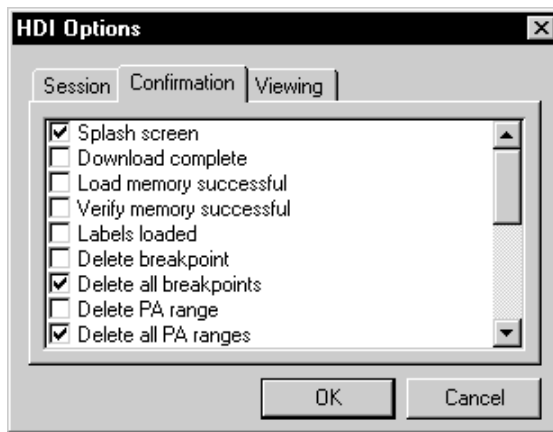


**Figure 15.11 HDI Options (Session) Dialog Box**

The 'On Exit' group of radio buttons automates saving the current session when the user program is shut down:

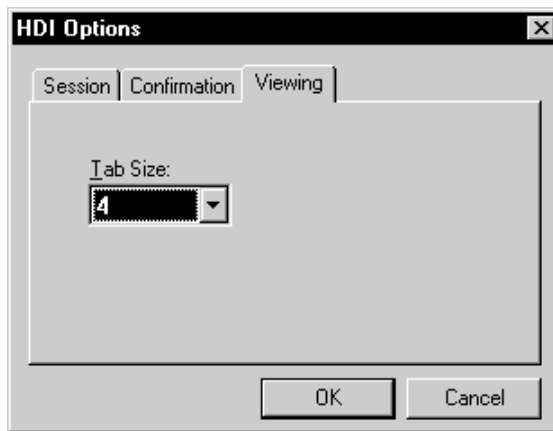
- **Save session automatically:** This will save the session information in the current session file. If there is no current session file then you will be prompted to enter an HDI session file name.
- **Prompt for save session:** This will always ask you if you want to save the current session when the program is shut down. If you select 'Yes', then the session information is saved in the current session file. If there is no current session file then you will be prompted to enter a session file name.
- **Quit without asking:** This shuts down the program and does not prompt you, nor save the current session information.

Check the 'Load last session on startup' check box if you want to automatically load the last saved session the next time the user program is started.



**Figure 15.12 HDI Options (Confirmation) Dialog Box**

Confirmation message boxes can be switched off or on by using the appropriate confirmation check box.



**Figure 15.13 HDI Options (Viewing) Dialog Box**

The 'Tab Size' list box can be used to set the number of spaces that a tab character will be expanded to within the views. Valid values are between 2 and 8. The best value will be the same as your normal editor.

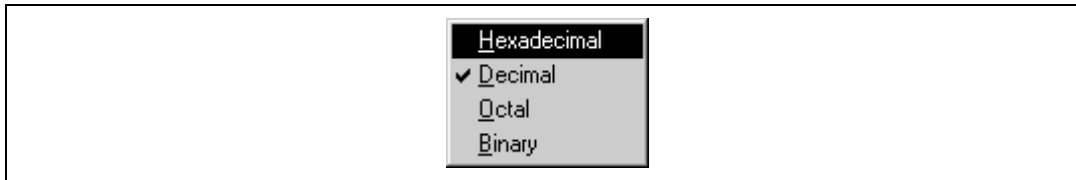
## 15.10 Setting the Default Input Radix

HDI can accept input in several numerical bases. The default is hexadecimal (except Count fields which are always decimal), but you can also use one of the prefixes described in section 2.2.2,



Data Formats. To improve usability, you can select one of these formats as the default, i.e. you will not need to enter the corresponding prefix to use that radix.

To change the default radix, choose the [**Setup->Radix**] menu option. This will display a list of possible numbering systems with a check mark to the left of the current radix:

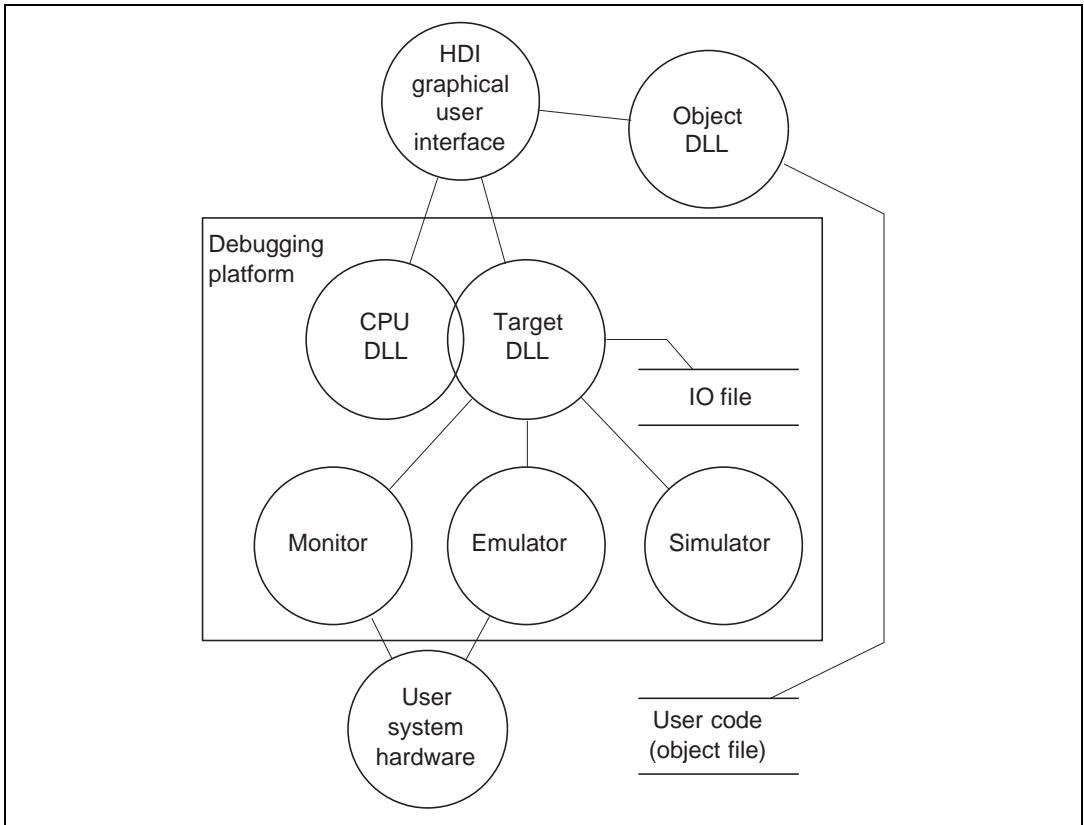


**Figure 15.14 Setting Radix**



# Appendix A - System Modules

The following section describes the architecture of the HDI debugging system.



**Figure A.1 HDI System Modules**

In normal operation, the user program will be placed directly into the user system hardware (for example as an EPROM). HDI uses this information to provide a Windows<sup>®</sup>-based debugging system.

To decrease the learning curve when swapping between different debugging platforms and/or user system hardware, HDI provides a single unified interface (the GUI) and a family of target specific modules. Normally, the user will only interact with the standard GUI - once the appropriate target module has been selected, the rest of the system configures itself automatically by loading the appropriate modules.

## Graphical User Interface

This is the main HDI.EXE program that runs under Windows®. It uses familiar Windows® operations, with menus and windows to give a user-friendly view into the debugging system. The GUI is the only contact between the user and the rest of the system; it processes commands and provides the required information about the user program. It also provides the interface between the module DLLs and the host file system, i.e., the PC.

## **Object DLL**

When creating the user program, a compiler will generate an *absolute object file*. This file contains the actual machine code and data that the microcomputer processes to execute the functions making up the target application. In order to debug the user program as original source code, the compiler must provide more information to the debugger. For this reason, nearly all compilers have a debugging option that puts all the information necessary for debugging your source code into the absolute file, which is usually called a debug object file.

The object DLL extracts this information from the object file for display to the user. Since the format of data is compiler dependent, more than one object DLL may be present in the HDI directory - HDI will try each in turn until it finds one that can understand the object file's format.

## **CPU DLL**























The CPU DLL module contains information specific to the target microcomputer. For example, it contains the number and types of registers available to the microcomputer. It also translates the raw machine code in the target into more familiar assembly-language mnemonics displayed in the **Source** window, and vice versa.
























## **Target DLL**













The target DLL informs HDI about the debugging platform's capabilities and selects the correct CPU DLL. Since some capabilities of the debugging platform cannot be generic (for example, target configuration), the target DLL also includes extensions to the standard GUI to provide the user with access to these capabilities.

For a detailed description of the features available using your target DLL, refer to the supplied debugging platform user's manual.

# Appendix B - GUI Command Summary

Menu	Item	Accelerator	Toolbar Graphic
File	New Session...	Ctrl+N	
	Load Session...	Ctrl+O	
	Save Session	Ctrl+S	
	Save Session As...		
	Load Program...		
	Initialize		
	Exit	Alt+F4	
Edit	Cu <u>t</u>	Ctrl+X	
	<u>C</u> opy	Ctrl+C	
	<u>P</u> aste	Ctrl+V	
	<u>F</u> ind	F3	
	<u>E</u> valuate		
View	<u>B</u> reakpoint	Ctrl+B	
	<u>C</u> ommand Line	Ctrl+L	
	<u>D</u> isassembly...	Ctrl+D	
	<u>I</u> /O Area	Ctrl+I	
	<u>L</u> abels	Ctrl+A	
	<u>L</u> ocals	Ctrl+Shift+W	
	<u>M</u> emory...	Ctrl+M	
	<u>P</u> erformance Analysis	Ctrl+P	
	<u>P</u> rofile-List	Ctrl+P	
	<u>P</u> rofile-Tree	Ctrl+F	
	<u>R</u> egisters	Ctrl+R	
	<u>S</u> ource...	Ctrl+K	

Menu	Item	Accelerator	Toolbar Graphic
View (cont)	Status	Ctrl+U	
	Trace	Ctrl+T	
	Watch	Ctrl+W	
Run	Reset CPU		
	Go	F5	
	Reset Go	Shift+F5	
	Go To Cursor		
	Set PC To Cursor		
	Run...		
	Step In	F8	
	Step Over	F7	
	Step Out		
	Step...		
	Halt	Esc	
Memory	Refresh	F12	
	Load...		
	Save...		
	Verify...		
	Test...		
	Fill...		
	Copy...		
	Compare...		
	Search...		
	Configure Map...		
	Configure Overlay...		

Menu	Item	Accelerator	Toolbar Graphic
Setup	Options...		
	Radix (Input)		
	Hexadecimal		
	Decimal		
	Octal		
	Binary		
	Customize		
Toolbar...			
Font...			
File Filter...			
<hr/>			
	Configure Platform...		
Window	Cascade		
	Tile		
	Arrange Icons		
	<hr/>		
	Close All		
Help	Index	F1	
	Using Help		
	Search for Help on		
	<hr/>		
	About HDI		





# Appendix C - Symbol File Format

In order for HDI to be able to understand and decode the symbol file correctly, the file must be formatted in a specified manner:

1. The file must be a plain ASCII text file.
2. The file must start with the word “BEGIN”.
3. Each symbol must be on a separate line with the value first, in hexadecimal terminated by an “H”, followed by a space then the symbol text.
4. The file must end with the word “END”.

Example:

```
BEGIN
11FAH Symbol_name_1
11FCH Symbol_name_2
11FEH Symbol_name_3
1200H Symbol_name_4
END
```

---

## **H8S, H8/300 Series Simulator Debugger User's Manual**

Publication Date: 1st Edition, March 1994

6th Edition, September 2000

Published by: Electronic Devices Sales & Marketing Group  
Semiconductor & Integrated Circuits  
Hitachi, Ltd.

Edited by: Technical Documentation Group  
Hitachi Kodaira Semiconductor Co., Ltd.

Copyright © Hitachi, Ltd., 1994. All rights reserved. Printed in Japan.