To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

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.

User's Manual

RENESAS

HI2000/3 Renesas Industrial Realtime Operating System for H8S Series

User's Manual

Renesas Microcomputer Development Environment System

Renesas Electronics
www.renesas.com

Rev.2.0 2001.03

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.

1. μITRON is an acronym of the "Micro Industrial TRON" and TRON is an acronym of "The Realtime Operating system Nucleus".
2. Microsoft® Windows® 95, Microsoft® Windows® 98, and Microsoft® Windows NT® operating systems are registered trademarks of Microsoft Corporation in the United States and/or other countries.
3. All other product names are trademarks or registered trademarks of the respective holders.
4. This manual assumes the operating environment to be the English version of Microsoft® Windows® 95, Windows®98, and WindowsNT® operating system.

Preface

This manual describes how to configure systems using the Hitachi Industrial Realtime Operating System, a machine-installed realtime multitasking operating system based on μITRON3.0 specifications.

Please read this manual and the related manuals listed below before using the HI2000/3 to fully understand the operating system.

This user's manual contains the following eight sections and appendixes:

Section 1	Introduction to HI2000/3: general description of HI2000/3 systems
Section 2	Kernel: overview of HI2000/3 kernel functions. Refer to this section when creating the functions of the user system.
Section 3	System calls: overview of HI2000/3 kernel system calls. Refer to this section when creating the details of the user program and at coding.
Section 4	Functions and operations of the HI2000/3 debugging extension (DX)
Section 5	Handlers and routines: creating and defining handlers and routines necessary for HI2000/3 system configuration
Section 6	Setup table: creating the setup table required for HI2000/3 system configuration
Section 7	Interrupt vector table: creating the interrupt vector table required for HI2000/3 system configuration
Section 8	Load module: creating a load module
Appendixes	User and kernel work area calculation, description on compiler and assembler options, example of timer driver, a list of error codes, and a list of system call function codes.

The following shows the related manuals:

- HI2000/3 Release Notes
- H8S, H8/300 Series C/C++ Compiler User's Manual
- H8S, H8/300 Series Cross Assembler User's Manual
- H Series Linkage Editor, Librarian, and Object Converter User's Manual
- Hitachi Debugging Interface User's Manual
- Hitachi Integration Manager User's Manual
- The hardware manual and programming manual of the H8S microcomputer used

Symbols used in this manual have the following meanings:

[ ]: Parameters enclosed by [ ] can be omitted

(||): One of parameters enclosed by ( ) must be chosen

(RET): Pressing the RETURN key

Underlining (___): Indicates user's key input

< >: Contents shown in < > are to be specified

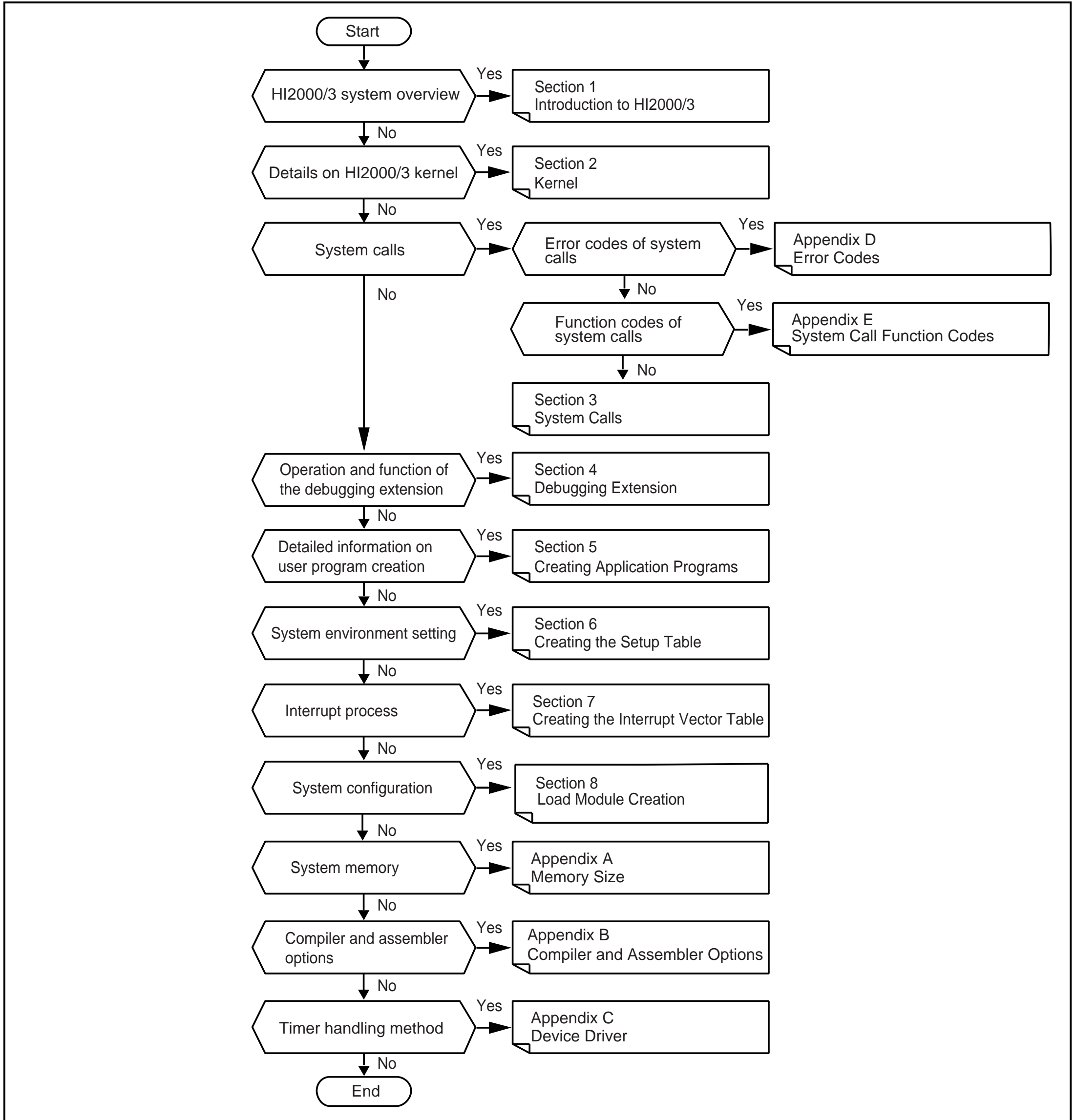
...: The entry specified just before this symbol can be repeated

H': For hexadecimal integers, prefix H' is attached. Default is D' (decimal).

nnnn: Bold-faced-italic ***nnnn*** is the device name.
Example: When the H8S/2655 is used, ***nnnn*** = 2655.

z: Bold-faced-italic ***z*** indicates the endian.
a is advanced mode.
n is normal mode.

It is recommended that the user refers to the following chart to understand the manual before reading.



Contents

Section 1 Introduction to HI2000/3 1
1.1 Overview 1
1.2 Features 1

Section 2 Kernel 3
2.1 Overview 3
2.2 Functions 3
2.3 System State 4
2.4 Tasks 6
2.4.1 Overview 6
2.4.2 Task State and Transition 7
2.4.3 Task Initiation 8
2.4.4 Task Scheduling 9
2.4.5 Task Waiting/Suspension and Release 9
2.4.6 Task Termination 11
2.4.7 Shared Stack Function 11
2.5 Synchronization and Communication 13
2.5.1 Event Flag 14
2.5.2 Semaphore 16
2.5.3 Mailbox 17
2.6 Interrupt 18
2.6.1 Overview 18
2.6.2 Interrupt Handler 18
2.6.3 Undefined Interrupt 18
2.6.4 Monopolizing the CPU 19
2.7 Memory Pool 19
2.7.1 Fixed-Size Memory Pool 20
2.7.2 Variable-Size Memory Pool 20
2.8 Time Management 22
2.8.1 Overview 22
2.8.2 Hardware Timer and System Clock 22
2.8.3 Setting and Referring to System Clock 22
2.8.4 Cyclic Handler 23
2.9 System Management 24
2.10 System-Call Trace 24
2.11 Trace Buffer Structure 25
2.12 Trace Acquisition Data Analysis Example 30
2.13 Trace-Function Definition 34
2.14 Notes on Trace Function 34

Section 3 System Calls 35
3.1 Overview 35
3.2 System Call Interface 36
3.2.1 C-Language Interface 36
3.2.2 Assembler Interface 39
3.2.3 Error Codes 40
3.3 System Calls 41
3.4 Task Management 42
3.4.1 Start Task (sta_tsk) [T/D/L] Start Task (ista_tsk) [D/I] 44
3.4.2 Exit Task (ext_tsk) [T/D/L] 46
3.4.3 Terminate Task (ter_tsk) [T/D/L] 48
3.4.4 Change Task Priority (chg_pri) [T/D/L] 50
3.4.5 Rotate Ready Queue (rot_rdq) [T/D/L] Rotate Ready Queue (irot_rdq) [D/I] 52
3.4.6 Release Wait (rel_wai) [T/D/L] 54
3.4.7 Get Task Identifier (get_tid) [T/D/L] 56
3.4.8 Refer Task State (ref_tsk) [T/D/L/I] 58
3.4.9 Disable Dispatch (dis_dsp) [T/D] 62
3.4.10 Enable Dispatch (ena_dsp) [T/D] 64
3.5 Task Synchronization 65
3.5.1 Suspend Task (sus_tsk) [T/D/L] 67
3.5.2 Resume Task (rsm_tsk) [T/D/L] 69
3.5.3 Sleep Task (slp_tsk) [T] Sleep Task with Timeout (tslp_tsk) [T] 71
3.5.4 Wakeup Task (wup_tsk) [T/D/L] Wakeup Task (iwup_tsk) [D/I] 73
3.5.5 Cancel Wakeup Task (can_wup) [T/D/L] 75
3.6 Synchronization and Communication (Event Flag) 77
3.6.1 Set Event Flag (set_flg) [T/D/L] Set Event Flag (iset_flg) [D/I] 79
3.6.2 Clear Event Flag (clr_flg) [T/D/L/I] 81
3.6.3 Wait for Eventflag (wai_flg) [T] Wait for Eventflag (Polling) (pol_fig) [T/D/L/I] Wait for Eventflag with Timeout (twai_fig) [T] 83
3.6.4 Refer Event Flag State (ref_flg) [T/D/L/I] 87
3.7 Synchronization and Communication (Semaphore) 89
3.7.1 Returns Semaphore Resource (sig_sem) [T/D/L] Returns Semaphore Resource (isig_sem) [D/I] 91
3.7.2 Wait on Semaphore (wai_sem) [T] Poll and Request Semaphore (preq_sem) [T/D/L/I] Wait on Semaphore with Timeout (twai_sem) [T] 93
3.7.3 Refer Semaphore State (ref_sem) [T/D/L/I] 96
3.8 Synchronization and Communication (Mailbox) 98
3.8.1 Send Message to Mailbox (snd_msg) [T/D/L] Send Message to Mailbox (isnd_msg) [D/I] 100
3.8.2 Receive Message from Mailbox (rcv_msg) [T] Poll and Receive Message from Mailbox (prcv_msg) [T/D/L/I] Receive Message from Mailbox with Timeout (trcv_msg) [T] 103
3.8.3 Refer Mailbox Status (ref_mbx) [T/D/L/I] 106

3.9 Interrupt Management 108
3.9.1 Return from Interrupt Handler (ret_int) [I] 111
3.9.2 Change Interrupt Mask Level (chg_ims) [T/I] 112
3.9.3 Refer Interrupt Mask Level State (ref_ims) [T/D/L/I] 114
3.9.4 Lock CPU (loc_cpu) [T/D/L] 116
3.9.5 Unlock CPU (unl_cpu) [T/D/L] 119
3.10 Memory Pool Management (Fixed-Size Memory Pool) 121
3.10.1 Get Fixed-Size Memory Block (get_blf) [T] Poll and Get Fixed-Size Memory Block (pget_blf) [T/D/L/I] Get Fixed-Size Memory Block with Timeout (tget_blf) [T] 123
3.10.2 Release Fixed-Size Memory Block (rel_blf) [T/D/L] 126
3.10.3 Refer Fixed-Size Memory Pool Status (ref_mpf) [T/D/L/I] 128
3.11 Memory Pool Management (Variable-Size Memory Pool) 131
3.11.1 Get Variable-Size Memory Block (get_blk) [T] Poll and Get Variable-Size Memory Block (pget_blk) [T/D/L/I] Get Variable-Size Memory Block with Timeout (tget_blk) [T] 133
3.11.2 Release Variable-Size Memory Block (rel_blk) [T/D/L] 137
3.11.3 Refer Variable-Size Memory Pool Status (ref_mpl) [T/D/L/I] 139
3.12 Time Management 142
3.12.1 Set System Clock (set_tim) [T/D/L/I] 144
3.12.2 Get System Clock (get_tim) [T/D/L/I] 146
3.12.3 Activate Cyclic Handler (act_cyc) [T/D/L/I] 148
3.12.4 Refer Cyclic Handler State (ref_cyc) [T/D/L/I] 150
3.13 System Management 153
3.13.1 get_ver (Get Version Information) [T/D/L/I] 153

Section 4 Debugging Extension 157
4.1 Overview 157
4.1.1 Displaying and Manipulating Objects 157
4.1.2 Results of Object Manipulation 159
4.1.3 Displaying the Register Values 160
4.1.4 Displaying the HI2000/3 DX System Call Trace Information 161
4.1.5 Online Help 161
4.2 List of Functions 162
4.2.1 HI2000/3 DX Menus 162
4.2.2 Windows and Dialog Boxes 163
4.3 Notes 164
4.3.1 Setting up the E6000 Emulator 164
4.3.2 Displaying the HI2000/3 DX Window 164
4.3.3 Realtime Operation of the User System 164
4.3.4 Displaying Correct Data 164
4.3.5 Trace 165
4.3.6 User System Memory 165

4.3.7 Correspondence to the HDI Session ... 165
4.3.8 Loading Load Modules ... 165
4.4 Debug Daemon ... 166
4.5 Tutorial ... 167
4.5.1 Executing a Sample Program ... 168
4.5.2 Starting a Task ... 171
4.5.3 Mailboxes and Messages ... 173
4.5.4 Examples during System Operation ... 175

Section 5 Creating Application Programs ... 179
5.1 Creating a User Program ... 179
5.2 Tasks ... 180
5.2.1 Creating Tasks ... 180
5.2.2 Defining Tasks ... 183
5.3 Interrupt Handlers ... 183
5.3.1 Interrupt Handler Description ... 183
5.3.2 Defining Interrupt Handlers ... 188
5.4 Undefined Interrupt Handlers ... 188
5.4.1 Creating Undefined Interrupt Handlers ... 188
5.4.2 Defining Undefined Interrupt Handlers ... 188
5.5 Cyclic Handlers ... 189
5.5.1 Creating Cyclic Handlers ... 189
5.5.2 Defining Cyclic Handlers ... 192
5.6 CPU Initialization Routine ... 192
5.6.1 Creating CPU Initialization Routines ... 192
5.6.2 Defining CPU Initialization Routines ... 194
5.7 Timer Initialization Routine ... 195
5.8 System Initialization Handlers ... 195
5.8.1 Creating System Initialization Handlers ... 195
5.8.2 Defining the System Initialization Handler ... 197
5.9 System Termination Routines ... 198
5.9.1 Creating System Termination Routines ... 198
5.9.2 Defining the System Termination Routine ... 202
5.10 System Idling Routine ... 202
5.10.1 Creating System Idling Routines ... 202
5.10.2 Defining a System Idling Routine ... 203

Section 6 Creating the Setup Table ... 205
6.1 Overview ... 205
6.2 User Definition Field ... 205
6.2.1 Defining the Constant Definition Field ... 206
6.2.2 Defining Task ... 210
6.2.3 Defining Fixed-Size Memory Pools ... 213

6.2.4 Defining Variable-Size Memory Pools 216
6.2.5 Defining Cyclic Handlers 218
6.2.6 Defining Trace Functions 221
6.2.7 Defining Extended Information 223
6.3 System Definition Field 229

Section 7 Creating the Interrupt Vector Table 231
7.1 Overview 231
7.2 Defining Interrupt Handler 231

Section 8 Load Module Creation 237
8.1 Overview 237
8.2 Workspace and Project Files 238
8.3 Load Module Creation 240
8.3.1 Adding Files to a Project 240
8.3.2 Compiler and Assembler Options 242
8.3.3 Inter-Module Optimizer Setting 249
8.3.4 Build Execution 255
8.4 C-Language Interface Library Projects 256

Appendix A Memory Size 257
A.1 Memory Size 257
A.1.1 OS Work Area Size Calculation 257
A.1.2 OS Stack Area Size Calculation 259
A.1.3 Timer Interrupt Stack Area Size Calculation 260
A.1.4 Task Stack Area Size Calculation 261
A.1.5 Interrupt Handler Stack Area Size Calculation 262
A.1.6 Fixed-Size Memory Pool Area Size Calculation 263
A.1.7 Variable-Size Memory Pool Area Size Calculation 263
A.1.8 Trace Function Stack Area Size Calculation 264
A.1.9 Trace Buffer Area Size Calculation 264
A.1.10 HI2000/3 Work Area Size Calculation 265

Appendix B Compiler and Assembler Options 267
B.1 Compiler Options 267
B.2 Assembler Options 267

Appendix C Device Driver 269
C.1 Timer Driver 269
C.1.1 Timer Initialization Routine 270
C.1.2 Timer Interrupt Handler 270
C.1.3 Timer Driver Definition Information 273

Appendix D Error Codes 277
D.1 System Call Error Codes 277
D.2 Debugging Extension Errors 278

Appendix E System Call Function Codes 281
E.1 System Call Function Codes 281

Figures Contents

Figure 2.1 System States....4
Figure 2.2 Task State Transition Diagram....8
Figure 2.3 Task State Transition when Using the Shared Stack Function....12
Figure 2.4 Example of Using an Event Flag....14
Figure 2.5 Exclusive Control of Resources by Semaphore....16
Figure 2.6 Mailbox Process....17
Figure 2.7 Fixed-Size Memory Pool Operation....20
Figure 2.8 Variable-Size Memory Pool Operation....21
Figure 2.9 Overview of Cyclic Handler Processing....23
Figure 2.10 Trace Buffer Structure....25
Figure 2.11 Trace Buffer Management Table Structure....25
Figure 2.12 Trace Buffer Management Process....26
Figure 2.13 Trace Entry Structure....27
Figure 2.14 Example of Trace Analysis Results....33
Figure 3.1 System Call Description Form....41
Figure 3.2 Message Form....102
Figure 4.1 Example of the Display of an Object (List-Type Window)....158
Figure 4.2 Example of the Display of an Object (Hierarchical-Type Window)....158
Figure 4.3 Example of Requesting Object Manipulation....159
Figure 4.4 [Task Context Registers] Window....160
Figure 4.5 [System Trace] Window....161
Figure 4.6 Sample Program Processing....167
Figure 4.7 HDI Initial Display....168
Figure 4.8 [Open] Dialog Box....169
Figure 4.9 [Tasks] Window....170
Figure 4.10 Source Code Display....171
Figure 4.11 Invoking Task....172
Figure 4.12 [System Trace] Window....172
Figure 4.13 [Mailboxes] Window....173
Figure 4.14 Step-Over Execution of Program....173
Figure 4.15 [Mailboxes] Window (Confirmation of Result)....174
Figure 4.16 [Mailboxes] Window (Expanded Display)....174
Figure 4.17 [System Trace] Window....174
Figure 4.18 [Tasks] Window after the [Update] Option is Selected....175
Figure 4.19 [Mailboxes] Window after the [Update] Option is Selected....175
Figure 4.20 [System Trace] Window....176
Figure 4.21 [Modify Task Status] Dialog Box....176
Figure 4.22 [Tasks] Window after the [Update] Option is Selected....177
Figure 4.23 [System Trace] Window....177
Figure 5.1 Kernel Initial Processing....179

Figures 5.2 Task Example in C Language 180
Figure 5.3 Interrupt Handler Example in C Language 184
Figure 5.4 Relationship between the Vector Table and the Interrupt Handler 188
Figure 5.5 Cyclic Handler Example for C Language 190
Figure 5.6 CPU Initialization Routine Example 193
Figure 5.7 System Initialization Handler Written in C Language 196
Figure 5.8 Stack State of the System Termination Routine 198
Figure 6.1 OS Stack Area Calculation 207
Figure 6.2 Constant Definition Field 208
Figure 6.3 Task Definition Field 210
Figure 6.4 Fixed-Size Memory Pool Definition Field 214
Figure 6.5 Variable-size Memory Pool Definition Field 217
Figure 6.6 Definition Example of Cyclic Handler Definition Field 220
Figure 6.7 Trace Function Definition Field 222
Figure 6.8 Extended Information Definition Field 224
Figure 6.9 System Definition Field 230
Figure 7.1 Coding Example from the Interrupt Vector Table 2655avec.src 232
Figure 8.1 Creating a Load Module 237
Figure 8.2 Selecting a Project 239
Figure 8.3 Adding Files to the Project 241
Figure 8.4 CPU Tab Window in the H8S, H8/300 Assembler Options 244
Figure 8.5 Object Tab Window in the H8S, H8/300 Assembler Options 245
Figure 8.6 List Tab Window in the H8S, H8/300 Assembler Options 246
Figure 8.7 Source Tab Window in the H8S, H8/300 Assembler Options (Include file directories) 247
Figure 8.8 Source Tab Window in the H8S, H8/300 Assembler Options (Defines) 248
Figure 8.9 Object Tab Window in the H8S, H8/300 C Compiler Options 249
Figure 8.10 Inter-Module Optimizer Options Input Tab 250
Figure 8.11 Inter-Module Optimizer Options Output Tab 252
Figure 8.12 Inter-Module Optimizer Options Section Tab 253
Figure 8.13 Executing the Build 255
Figure C.1 Timer Driver Processing 269
Figure C.2 Stack State at Timer Interrupt Reset Processing Termination 272

Tables Contents

Table 2.1 Task-Management System Calls 6
Table 2.2 Task Synchronization System Calls 6
Table 2.3 Task Waiting/Suspension and Release 10
Table 2.4 System Calls for Task Event Flag Control 13
Table 2.5 System Calls for Semaphore Control 13
Table 2.6 System Calls for Mailbox Control 14
Table 2.7 System Calls for Interrupt Control 18
Table 2.8 System Calls for Fixed-Size Memory Pool Control 19
Table 2.9 System Calls for Variable-Size Memory Pool Control 19
Table 2.10 System Calls for System Clock 22
Table 2.11 System Calls for Cyclic Handler Control 23
Table 2.12 System Call for Kernel Version Acquisition 24
Table 2.13 Trace Entry Data Meanings 29
Table 2.14 Trace Acquisition Data Example 30
Table 3.1 System Call Classification 35
Table 3.2 Parameter Prefixes and Suffixes 37
Table 3.3 System Calls for Task Management 42
Table 3.4 Task-Management Specifications 42
Table 3.5 Causes of Task-Execution Waiting/Suspension and Release 43
Table 3.6 Task Synchronization System Calls 65
Table 3.7 Task Synchronization Specifications 65
Table 3.8 Causes of Task-Execution Waiting/Suspension and Release 66
Table 3.9 System Calls for Event Flag Control 77
Table 3.10 Event Flag Specifications 77
Table 3.11 Causes of Task-Execution Waiting and Release 78
Table 3.12 Wait Modes (wfmode) 85
Table 3.13 System Calls for Semaphore Control 89
Table 3.14 Semaphore Specifications 89
Table 3.15 Causes of Task-Execution Waiting and Release 90
Table 3.16 System Calls for Mailbox Control 98
Table 3.17 Mailbox Specifications 98
Table 3.18 Causes of Task-Execution Waiting and Release 99
Table 3.19 System Calls for Interrupt Management 108
Table 3.20 Interrupt Mask Level in Interrupt Control Mode 0 108
Table 3.21 Interrupt Mask Level in Interrupt Control Mode 1 108
Table 3.22 Interrupt Mask Level in Interrupt Control Mode 2 109
Table 3.23 Interrupt Mask Level in Interrupt Control Mode 3 110
Table 3.24 State Transition by Issuing dis_dsp, ena_dsp, loc_cpu, and unl_cpu 118
Table 3.25 System Calls for Fixed-Size Memory Pool Control 121
Table 3.26 Fixed-Size Memory Pool Specifications 121

Table 3.27 Causes of Task-Execution Waiting and Release 122
Table 3.28 System Calls for Variable-Size Memory Pool Control 131
Table 3.29 Variable-Size Memory Pool Specifications 131
Table 3.30 Causes of Task-Execution Waiting and Release 132
Table 3.31 System Calls Related to the System Clock Control 142
Table 3.32 System Calls for Cyclic Handler Control 142
Table 3.33 System Clock Specifications 142
Table 3.34 Cyclic Handler Specifications 142
Table 3.35 Handler Activation State (cycact) 149
Table 4.1 Menu Items Added to the HDI [View] Menu 157
Table 4.2 HI2000/3 DX Menus 162
Table 4.3 HI2000/3 DX Windows and Dialog Boxes 163
Table 4.4 Memory Size Used by the User System 165
Table 4.5 Description of Trace Contents 172
Table 5.1 Resources Initialized at Task Initiation 181
Table 5.2 Resources and System Calls 181
Table 5.3 Conditions for Interrupt Handler Processing 185
Table 5.4 Conditions for Cyclic Handler Processing 191
Table 5.5 Conditions for CPU Initialization Routine Processing 194
Table 5.6 Conditions for System Initialization Handler Processing 197
Table 5.7 System Termination Causes 199
Table 5.8 Invalid Setup Information 200
Table 6.1 Information Defined in Constant Definition Field 206
Table 6.2 Contents of Task Definition 210
Table 6.3 Contents of Fixed-Size Memory Pool Definitions 213
Table 6.4 Contents of Variable-Size Memory Pool Definitions 216
Table 6.5 Contents of Cyclic Handler Definitions 219
Table 7.1 Defined Interrupt Handlers 231
Table 8.1 Sample Projects 238
Table 8.2 Files Required for Project 240
Table 8.3 Compiler and Assembler Options 243
Table 8.4 Supplied Library File List 251
Table 8.5 List of Sections Included in the Provided Project Files 254
Table 8.6 C-Language Interface Projects 256
Table A.1 OS Work Area Size Calculation 257
Table A.2 OS Stack Area Size Calculation 259
Table A.3 Timer Interrupt Stack Area Size Calculation 260
Table A.4 Task Stack Area Size Calculation 261
Table A.5 Interrupt Handler Stack Area Size Calculation 262
Table A.6 Fixed-Size Memory Pool Area Size Calculation 263
Table A.7 Variable-Size Memory Pool Area Size Calculation 263
Table A.8 Trace Function Stack Area Size Calculation 264
Table A.9 Trace Buffer Area Size Calculation 264

Table A.10 HI2000/3 Work Area Size Calculation....265
Table C.1 Conditions for Timer Initialization Routine Processing....270
Table C.2 Conditions for Timer Interrupt Reset Processing....271
Table C.3 Definition of Assign Directive for Timer Driver....274
Table D.1 System Call Error Codes....277
Table D.2 Debugging Extension Error Messages....278
Table E.1 System Calls and Function Codes....281

Section 1 Introduction to HI2000/3

1.1 Overview

The importance and complexity of developing operating systems (OSs) have grown along with the ever increasing use of microcomputer systems in a wide variety of fields. In particular, realtime OSs have gained wide acceptance in industrial measurement and control systems. The HI2000/3 is a realtime multitasking OS used in the assembly of industrial equipment. It operates with the H8S series CPU. The HI2000/3 is based on μITRON specifications (ver. 3.0).

The HI2000/3 has a debugging extension (DX), which is a software debugging tool for the application programs.

The HI2000/3 debugging extension (DX) can be used by installing it in the Hitachi Debugging Interface (HDI) and in the HI2000/3 system.

1.2 Features

The HI2000/3 has the following features.

- High-speed operating kernel

  Optimized to enable high-speed processing by using the high-speed H8S series CPU instruction sets.

  This kernel supports all four interrupt control modes provided by the H8S series CPU.

  Two kernels for normal mode and for advanced mode are available depending on the H8S series CPU operating mode. The normal mode kernel runs in a maximum address space of 64 kbytes. The advanced mode kernel runs in a maximum address space of 16 Mbytes (total data space of 4 Gbytes including the address space dedicated to data).

  Realtime speed has been improved, for example, by not checking parameters within the kernel.

- A compact kernel whose functions can be selected optionally

  The kernel program size and kernel work area size are reduced to minimize the ROM and RAM size on the user system. When the kernel functional module used with the user program is specified in the setup table, the kernel is easily configured with a minimal module size.

- High level language

  By using Hitachi's compiler, tasks and interrupt handlers can be written in C language.

- Debugging extension

  The debugging extension displays the history of the HI2000/3 system calls issued, refers and modifies the states of objects such as tasks through windows and dialog boxes, and debugs multitasking applications through an HDI. The debugging extension also provides a Windows® context help system.

- Sample programs

  The following sample source programs are provided. By modifying the programs as required, the user system can be created easily.

  — System configuration files (such as the kernel-build file and the setup table)
  — Handlers and routines
  — Timer driver for H8S series on-chip Timer Pulse Unit (TPU) and Free Running Timer (FRT)
  — Task examples: Tutorial for HI2000/3 debugging extension (DX)

Section 2 Kernel

2.1 Overview

The kernel, which is the nucleus of the operating system HI2000/3, performs realtime multitasking processing. It has three major roles as follows:

- Response to events

  Recognizes events generated asynchronously, and immediately executes a task to process the event.

- Task scheduling

  Schedules task execution on a priority basis.

- System call execution

  Accepts various processing requests (system calls) from tasks and performs the appropriate processing.

2.2 Functions

Almost all kernel functions can be used by issuing system calls from an application program.

Task Management: When a task is executed, CPU is allocated to the task. The kernel controls the order of CPU allocation and starting and terminating tasks. Multiple tasks can share a stack by using the shared-stack function.

Task Synchronization Management: Performs basic synchronous processing for tasks, such as task execution suspension and release from other tasks, and performs synchronous processing between tasks.

Synchronization and Communication Management: Performs inter-task synchronization and communication by using event flags, semaphores, and mailboxes.

Interrupt Management: Initiates interrupt handlers in response to external interrupts. The interrupt handler performs appropriate interrupt processing, and notifies tasks of interrupt occurrences.

Memory Pool Management: Manages unused memory within the user system as a memory pool. A task acquires or returns memory blocks from the memory pool dynamically. Memory pools are either fixed-size memory pool or variable-size memory pool.

Time Management: Manages time-related information for the system and monitors task execution time for control purposes.

System Management: Reads the kernel version number.

System-Call Trace: Stores system call issuance history for system calls that are being executed.

2.3 System State

System states can be classified as shown in figure 2.1. When configuring a user system, the system state must be considered.

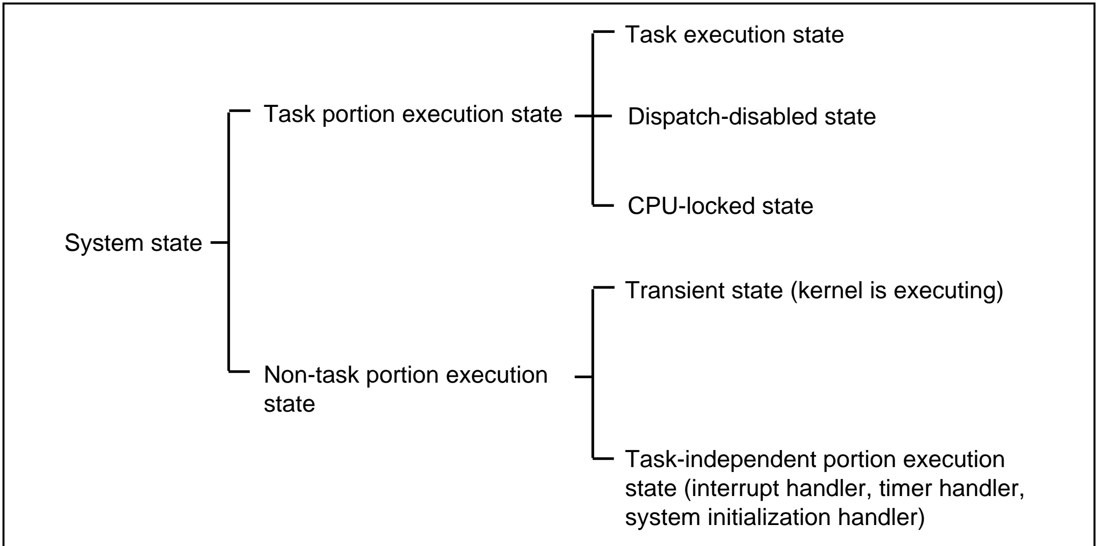


Figure 2.1 System States

The descriptions for the system states are given as follows.

Task Portion Execution State: A task is executing. The following are the three possible sub-states:

- Task-execution state

  The task portion is executing and allows task switching and interrupts. Tasks execute in this state.

  Tasks are not dispatched (scheduled) in any state other than this state. If the system is in a state other than this state, task scheduling is delayed until the system returns to this state.

  System calls that can be issued in the task-execution state can be used.

- Dispatch-disabled state

  The task portion is executing but does not allow task dispatch (scheduling).

  Issuing the dis_dsp system call while tasks are being executed disables task dispatch. Issuing the ena_dsp system call enables task dispatch again.

  In this state, system calls that shift a task to WAIT state cannot be used; only system calls that can be used while task dispatch is disabled can be used.

- CPU-locked state

  The task portion is executing but does not allow dispatches or interrupts.

  Issuing the loc_cpu system call while tasks are being executed locks the CPU. Issuing the unl_cpu unlocks the CPU.

  In this state, system calls that shift a task to a WAIT state cannot be used; only system calls that can be issued in the CPU-locked state can be used.

Non-Task Portion Execution State: Functions other than a task portion are executing. The following are the two possible sub-states.

- Transient state (while the kernel is under execution)

  The kernel is executing, that is, processing a system call.

- Task-independent portion execution state

  A feature of task-independent portions is that they do not recognize themselves as currently running processes because task-independent portions are completely independent of tasks. Therefore, in this portion, such a system call that specifies itself cannot be issued, e.g., a system call to put itself into the WAIT state. Tasks are not switched either; task switching is delayed until the system returns to the task-execution state.

  Possible task-independent portions are interrupt handlers, timer interrupt handlers, and system initialization handlers.

  System calls for the task-independent portion can be used in this state.

  Note that masking interrupts (changing the mask value from zero to another value) by issuing the chg_ims system call during task portion execution immediately moves the system from a task portion to a task-independent portion. Returning the interrupt mask value to zero returns the system to task portion execution.

2.4 Tasks

2.4.1 Overview

In a realtime multitasking system, the user prepares the application program in task units that can be processed independently and in parallel.

A task communicates with other tasks using system calls provided by the kernel. The kernel can process events that are asynchronously generated by external devices or the MCU through such system calls.

Tables 2.1 and 2.2 list the system calls that operate the tasks.

Table 2.1 Task-Management System Calls

System Call	Description
sta_tsk ista_tsk	Starts task
ext_tsk	Terminates current task
ter_tsk	Forcibly terminates a task
chg_pri	Changes task priority
rot_rdq irot_rdq	Rotates task ready queue
rel_wai	Releases the task WAIT state
get_tid	Refers current task ID
ref_tsk	Refers task state
dis_dsp	Disables dispatch
ena_dsp	Enables dispatch

Table 2.2 Task Synchronization System Calls

System Call	Description
sus_tsk	Shifts task to SUSPEND state
rsm_tsk	Resumes the execution of a task in SUSPEND state
slp_tsk	Shifts current task to WAIT state
tslp_tsk	Shifts current task to WAIT state (with timeout function)
wup_tsk iwup_tsk	Wakes up task
can_wup	Cancels wake-up task

2.4.2 Task State and Transition

A task enters one of following six states in the user system.

DORMANT State: A task has been registered in the kernel but has not yet been initiated, or has already been terminated.

READY (executable) State: An executable task is queuing to wait for CPU allocation because another task with a higher priority is currently running.

RUN State: The task is currently running.

WAIT State: A task is waiting for an event to occur. A task is placed in the WAIT state when, in the RUN state, it issues a system call to makes itself enter the WAIT state because its execution conditions are not satisfied. The task is placed in the READY state when it is released from the WAIT state.

SUSPEND State: A task has been suspended by another task.

WAIT-SUSPEND State: This state is a combination of both the WAIT state and the SUSPEND state.

Figure 2.2 shows the task state transition diagram.

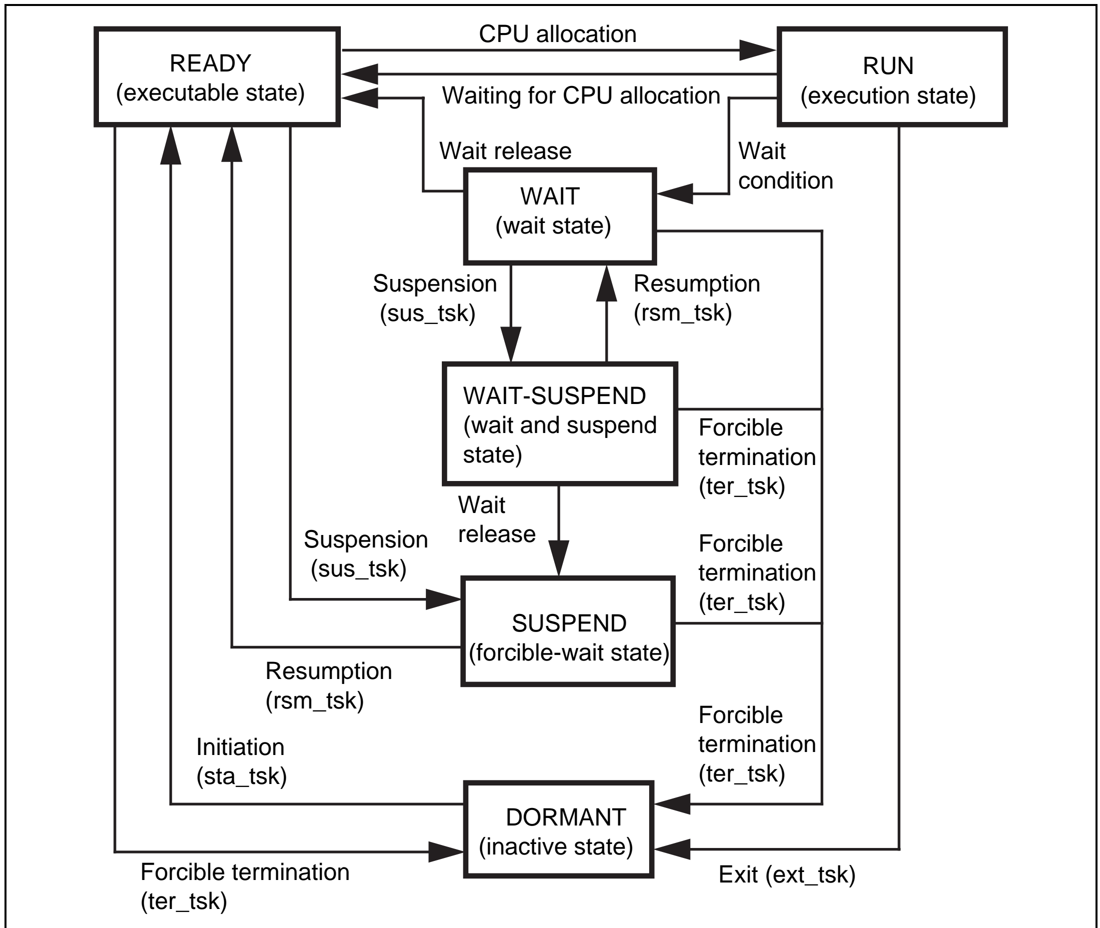


Figure 2.2 Task State Transition Diagram

2.4.3 Task Initiation

Task initiation means that a task in the DORMANT state makes a transition to the READY state. A task can be initiated by either of the following methods:

- Issuing the sta_tsk or ista_tsk system call to the target task
- Defining initial task initiation in the setup table

2.4.4 Task Scheduling

Task scheduling means that the kernel determines the order of execution for executable tasks, that is, the order of allocating the CPU to a task in the READY state. The kernel selects one task in the READY state and shifts it to the RUN state. If there are no tasks in the READY state, the kernel enters the idle state and waits for a task to be waken up via an interrupt. When there is more than one task in the READY state, the execution order is determined by using the CPU allocation wait queue, which is called the ready queue. There is a ready queue for each level of the maximum number of task priority, each queue operating on a first-come, first-served (FCFS) basis. The lower the number, the higher the priority is.

The kernel also supports round-robin scheduling, where the CPU allocates the same amount of time for each task with the same priority by rotating the ready queue at specific intervals. There are two types of scheduling: standard scheduling and round-robin scheduling. The round-robin scheduling manipulates the ready queues through the rot_rdq and irot_rdq system calls. Round-robin scheduling can be achieved by rotating the ready queue by issuing irot_rdq system call from the timer interrupt handler which is initiated at specific intervals.

2.4.5 Task Waiting/Suspension and Release

An executing task shifts to the WAIT or SUSPEND state when an interrupt occurs or a resource becomes unavailable; a task returns to the previous state when the cause of shifting a task to the WAIT state or SUSPEND state is cancelled. Note, however, a task does not always resume execution immediately after the cause of shifting a task to the WAIT or SUSPEND state is cancelled; actual execution timing is determined according to the event-driven scheduling. Table 2.3 lists the cause of shifting an executing task to the WAIT or SUSPEND state.

Table 2.3 Task Waiting/Suspension and Release

Cause of Waiting/Suspension		Time of Release
When the current task enters the WAIT state	slp_tsk or tslp_tsk system call	(1) When system call wup_tsk is issued (2) When the specified timeout period (tmout) has passed (tslp_tsk) (3) When system call rel_wai is issued
	wai_flg or twai_flg system call	(1) When the event-flag wait condition is satisfied (2) When the specified timeout period (tmout) has passed (twai_flg) (3) When system call rel_wai is issued
	wai_sem or twai_sem system call	(1) When the resource managed by semaphore is acquired (2) When the specified timeout period (tmout) has passed (twai_sem) (3) When system call rel_wai is issued
	rcv_msg or trcv_msg system call	(1) When a message is sent to the mailbox (2) When the specified timeout period (tmout) has passed (trcv_msg) (3) When system call rel_wai is issued
	get_blf to tget_blf system call	(1) When a memory block is acquired (2) When the specified timeout period (tmout) has passed (tget_blf) (3) When system call rel_wai is issued
	get_blk to tget_blk system call	(1) When a memory block is acquired (2) When the specified timeout period (tmout) has passed (tget_blk) (3) When system call rel_wai is issued
When forcibly suspended by another task	sus_tsk system call	When system call rsm_tsk is issued
When an interrupt is generated		When an interrupt handler completes execution
When a shared stack is being occupied	sta_tsk system call	When the shared stack is released

2.4.6 Task Termination

Task termination means that a task completes execution and enters the DORMANT state by one of the following methods:

- An ext_tsk system call is issued for the current task
- A ter_tsk system call is issued for the target task

Resources acquired with system calls must be returned before a task is terminated. Once a task is terminated, it is executed again from the initial state when initiated.

2.4.7 Shared Stack Function

More than one task can share one static stack area. This reduces the total stack area. A shared stack is defined in the setup table. However, only one task at a time can be executed in a task group that shares a stack.

A shared stack is released when the task using the shared stack enters the DORMANT state. If there is a task waiting for a shared stack, the task at the head of the shared-stack waiting queue uses the shared stack and enters the READY state.

The shared-stack wait queue is managed on a first-in first-out (FIFO) basis. The tasks are connected to the shared-stack wait queue in the order of the initiation request.

When tasks compete to use the same stack, the task that is initiated first uses the stack, and the other tasks wait for the shared stack.

Figure 2.3 shows the task state transition when using the shared stack function.

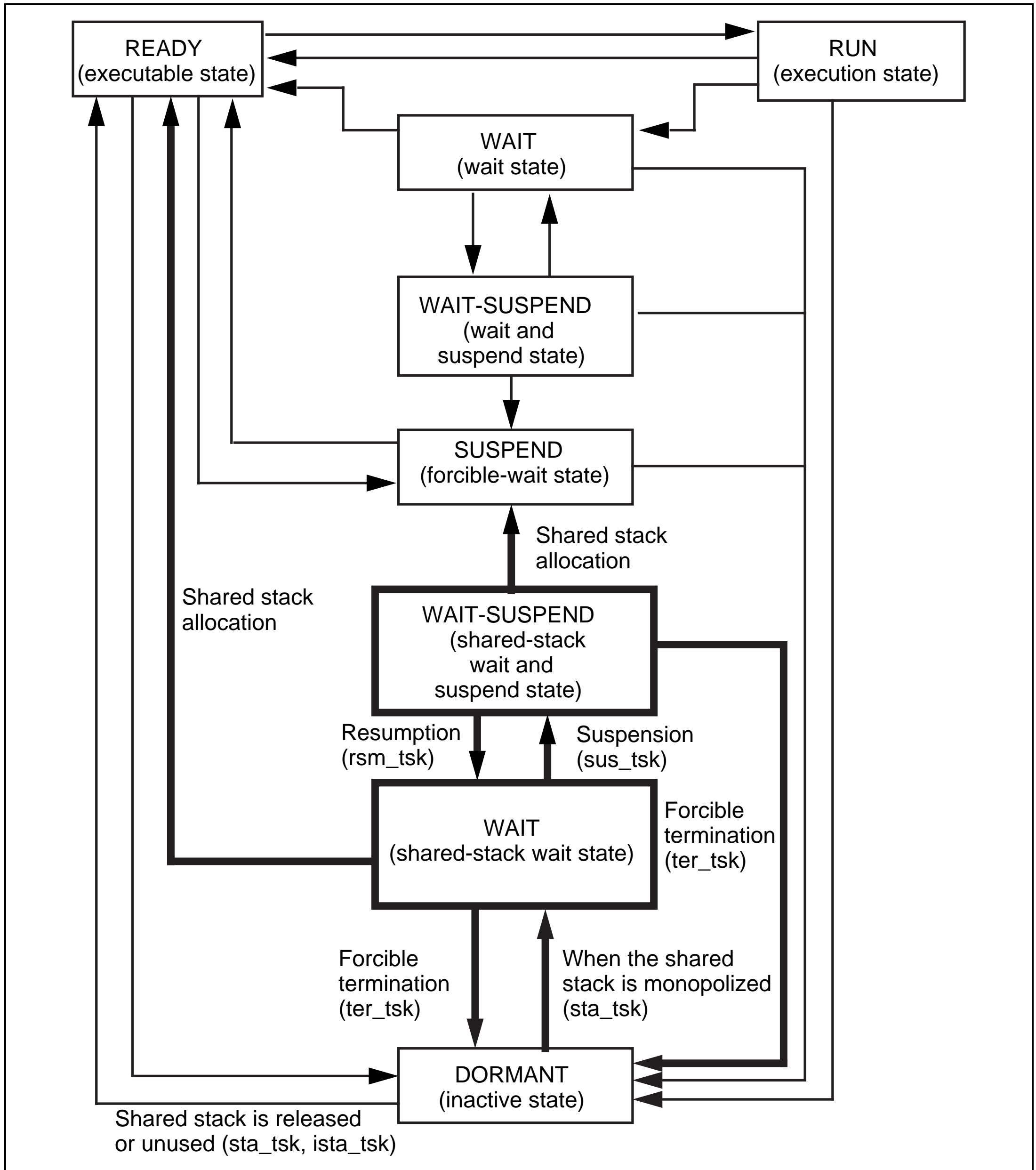


Figure 2.3 Task State Transition when Using the Shared Stack Function

2.5 Synchronization and Communication

For synchronization and communication purposes, the kernel has the following objects which are independent of tasks.

- Event flags
  Waits for several events and synchronizes task operations.
- Semaphores
  Exclusively controls resources.
- Mailboxes
  Transfers data (passes pointer to data).

The task event flags are controlled by the system calls listed in table 2.4.

Table 2.4 System Calls for Task Event Flag Control

System Call	Description
set_flg iset_flg	Sets event flag
clr_flg	Clears the event flag
wai_flg	Waits for event flag
pol_flg	Polls and gets event flag
twai_flg	Waits for event flag with timeout
ref_flg	Refers to the event flag state

Table 2.5 System Calls for Semaphore Control

System Call	Description
sig_sem isig_sem	Returns semaphore resource
wai_sem	Gets semaphore resource
preq_sem	Polls and gets semaphore resource
twai_sem	Gets semaphore resource with timeout
ref_sem	Refers to the semaphore state

Table 2.6 System Calls for Mailbox Control

System Call	Description
snd_msg isnd_msg	Sends message to mailbox
rcv_msg	Receives message from mailbox
prcv_msg	Polls and receives message from mailbox
trcv_msg	Receives message from mailbox with timeout
ref_mbx	Refers to the mailbox state

2.5.1 Event Flag

Event flags are used to enable quick inter-task synchronization by combining various events. An event flag is a bit-group corresponding to events. The value one represents event occurrence and zero represents no event occurrence. More than one task can wait for a specified bit to be set in an event flag, that is, tasks can wait until the specified event occurs.

Figure 2.4 shows an example of using event flags.

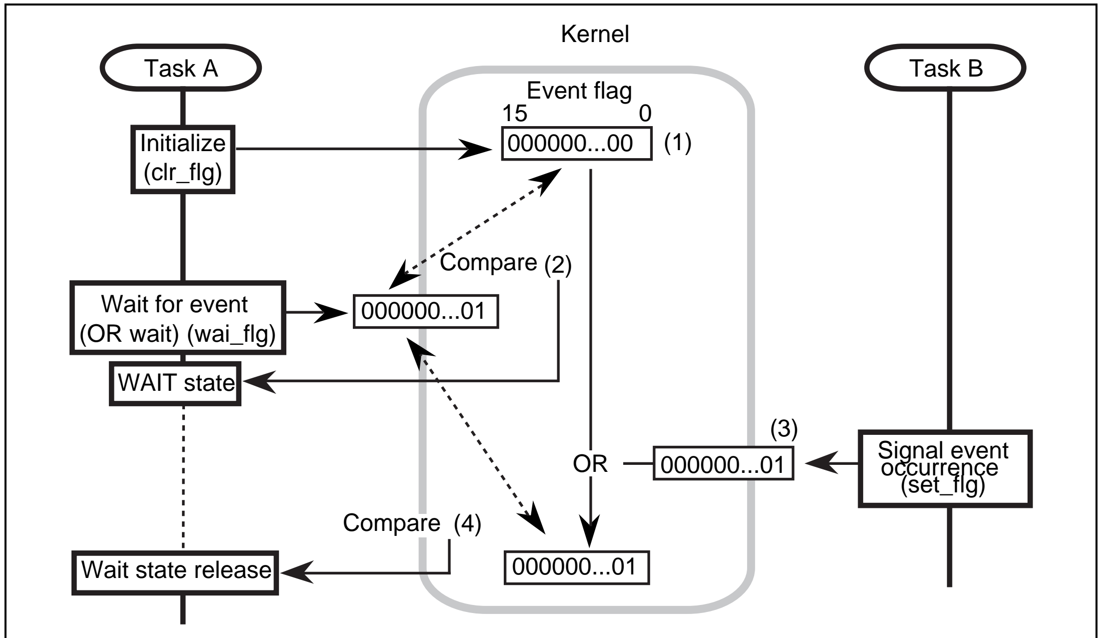


Figure 2.4 Example of Using an Event Flag

Description:

(1) Task A clears an event flag with the initial value specified.

(2) As the specified event has not occurred yet, task A waits for a specified event occurrence in the OR wait mode. (OR wait: to wait for at least one specified event to occur)

(3) Task B signals event occurrence; the bits of the event flag are then set.

(4) Task A is released from the WAIT state because the wait release condition is satisfied.

2.5.2 Semaphore

Elements such as I/O and shared memory required for task execution are called resources. Most resources are exclusively controlled by semaphores. Semaphores have non-negative counters that indicate the number of resources available. A task acquires semaphore counter values and can use resources corresponding to the counter values acquired. That is, the acquisition of semaphore counter values is the same as the acquisition of resources. Figure 2.5 shows an example of exclusive control of resources by the semaphore.

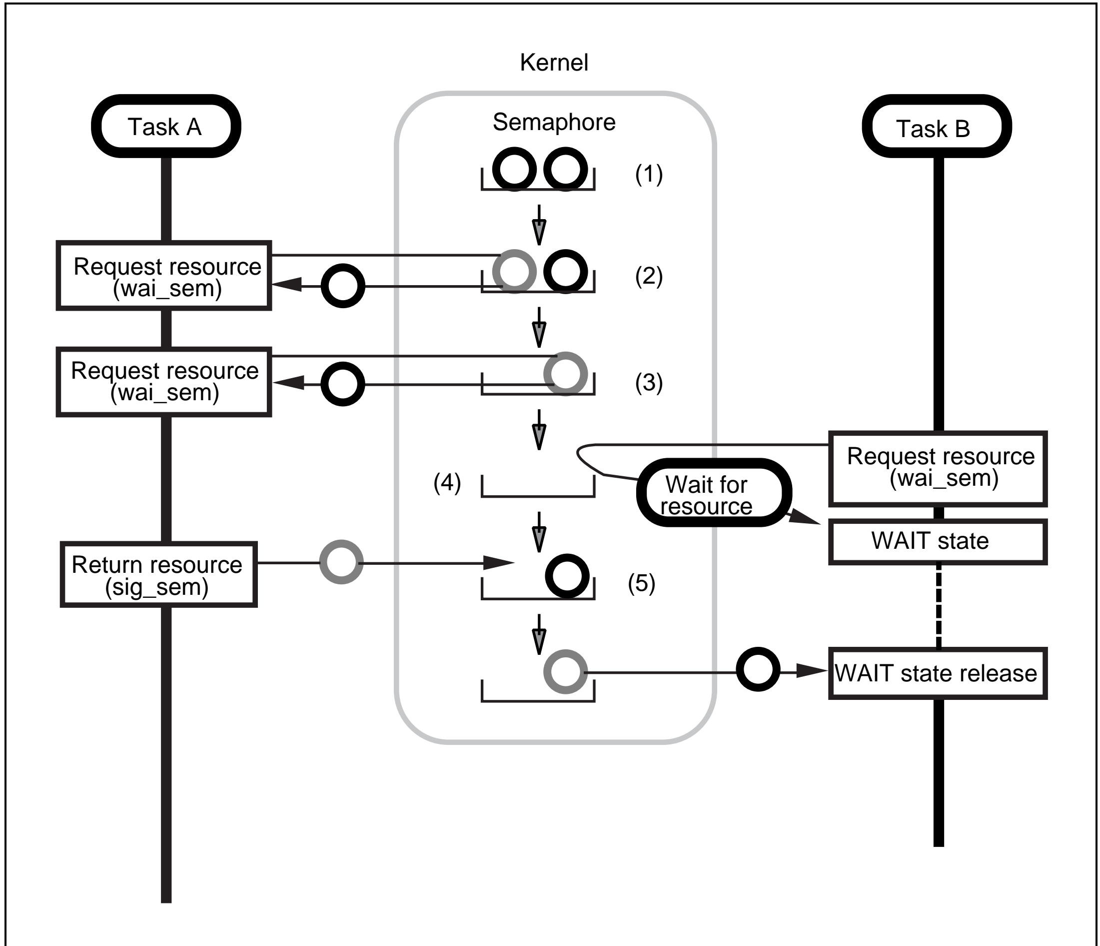


Figure 2.5 Exclusive Control of Resources by Semaphore

Description:

(1) First, two resources are set (semaphore counter = 2).

(2) Task A requests and gets a resource (semaphore counter = 1).

(3) Task A requests and gets another resource (semaphore counter = 0).

(4) Task B requests a resource, but has to enter the WAIT state because there is no resource.

(5) Task A returns a resource. The released resource is allocated to task B and task B is released from the WAIT state.

2.5.3 Mailbox

Mailboxes are used to when message data is sent and received between tasks. A message is sent to a mailbox from a task, and is later sent on to another task from the mailbox. The mailbox sends the start address of a message. Since the communication using a mailbox sends and receives the message start address, it is fast regardless of the message size.

Figure 2.6 shows the mailbox process.

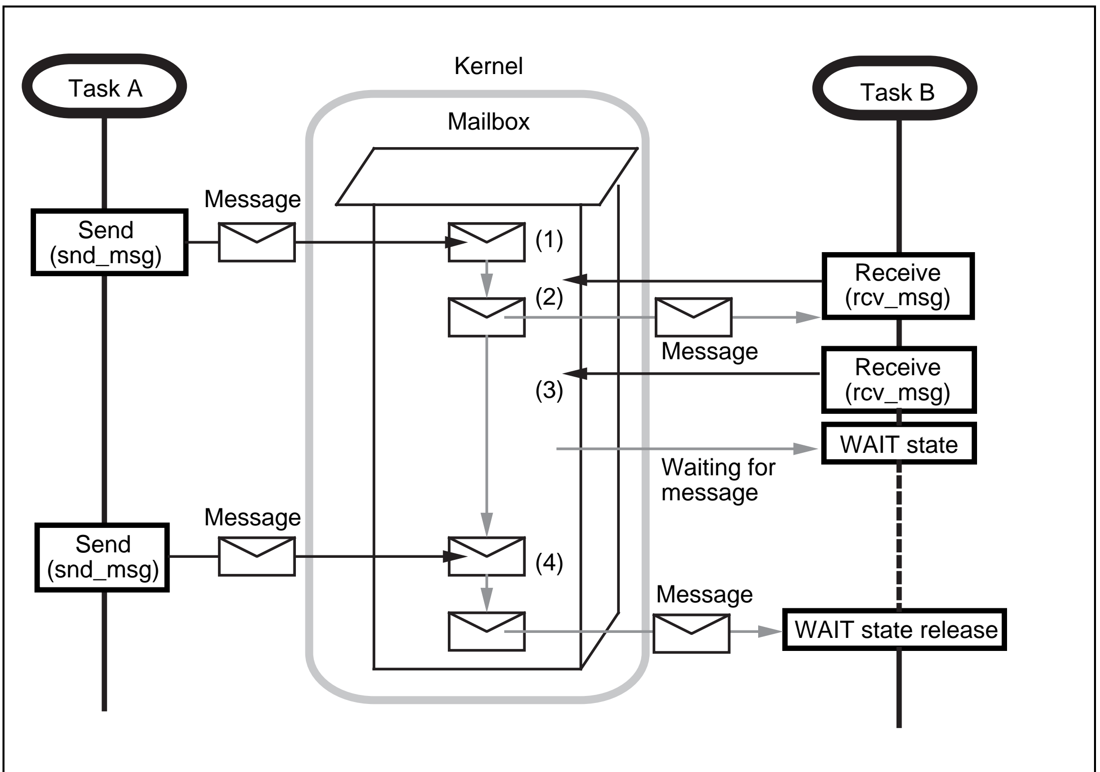


Figure 2.6 Mailbox Process

Description:

(1) Task A sends a message to the mailbox, storing one message in the mailbox.

(2) Task B issues a message receive request and the message is transferred to task B.

(3) Task B again issues a message receive request, but it is placed in the WAIT state since no message is in the mailbox.

(4) Task A sends a message, and task B is released from the WAIT state to receive the message.

2.6 Interrupt

2.6.1 Overview

When an interrupt occurs from an external hardware or a peripheral module, the interrupt handler is initiated without kernel intervention.

The interrupts are controlled by the system calls listed in table 2.7.

Table 2.7 System Calls for Interrupt Control

System Call Name	Function
ret_int	Returns from the interrupt handler
chg_ims	Changes the interrupt mask level
ref_ims	Refers to the interrupt mask level
loc_cpu	Disables interrupts and dispatches
unl_cpu	Enables interrupts and dispatches

2.6.2 Interrupt Handler

When an interrupt occurs, the currently running task is suspended until the interrupt handler completes execution.

Tasks are scheduled after the interrupt handler has completed execution; tasks are not scheduled even when a task with high priority is in the READY state due to the system call issued while the interrupt handler was being executed.

When interrupts are nested, the tasks are scheduled when all the interrupt handlers have completed execution.

2.6.3 Undefined Interrupt

If an undefined interrupt occurs, the system fails, passing the undefined interrupt or exception information as parameters to the system termination routine.

2.6.4 Monopolizing the CPU

A task can monopolize the CPU in two ways: One is to issue system call cpu_loc to lock the CPU. To unlock the CPU, system call unl_cpu must be issued. The other is to issue system call chg_ims to mask interrupts. If system call chg_ims is issued, the system makes a transition from the task portion execution to task-independent portion execution. During task-independent portion execution, only limited number of system calls can be issued and scheduling will be delayed as well as in the CPU-locked state.

2.7 Memory Pool

The memory pools allow memory space to be used efficiently. The HI2000/3 provides fixed-size memory pools and variable-size memory pools.

The fixed-size memory pools are controlled by the system calls listed in table 2.8.

Table 2.8 System Calls for Fixed-Size Memory Pool Control

System Call Name	Function
get_blf	Gets a fixed-size memory block
pget_blf	Polls and gets a fixed-size memory block
tget_blf	Gets a fixed-size memory block with timeout
rel_blf	Returns a fixed-size memory block
ref_mpf	Reads the fixed-size memory pool status

The variable-size memory pools are controlled by the system calls listed in table 2.9.

Table 2.9 System Calls for Variable-Size Memory Pool Control

System Call Name	Function
get_blk	Gets a variable-size memory block
pget_blk	Polls and gets a variable-size memory block
tget_blk	Gets a variable-size memory block with timeout
rel_blk	Returns a variable-size memory block
ref_mpl	Refers to the variable-size memory pool status

2.7.1 Fixed-Size Memory Pool

A fixed-size memory pool consists of fixed-size memory areas called memory blocks. A task can get a fixed-size memory block from the memory pool.

Figure 2.7 shows how the fixed-size memory pool works.

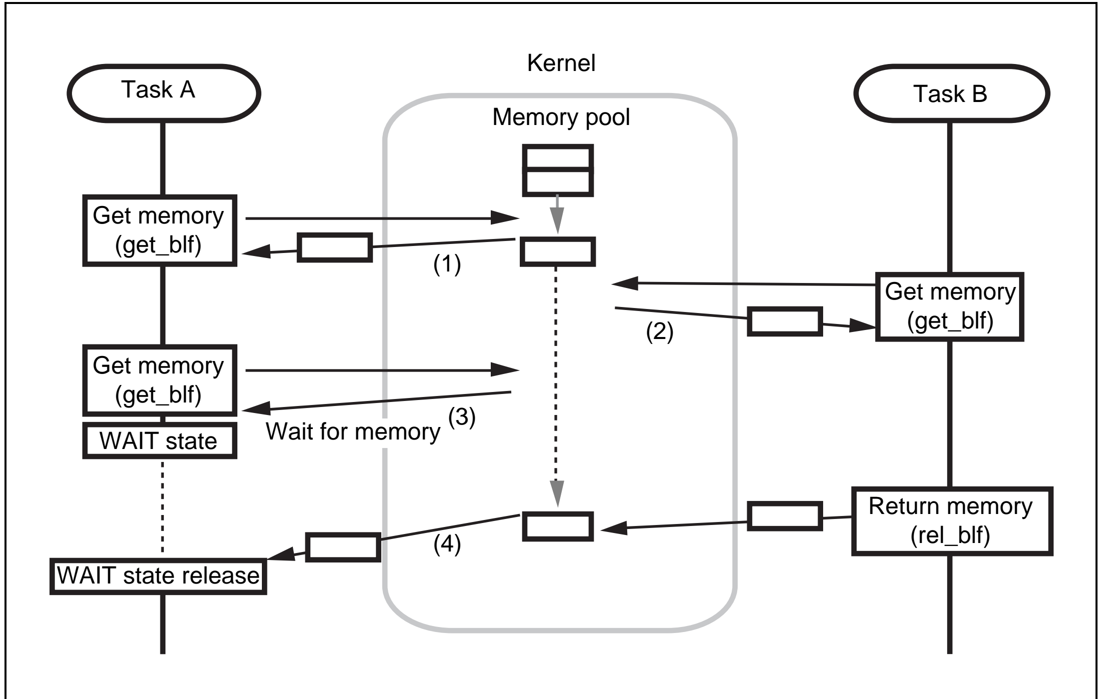


Figure 2.7 Fixed-Size Memory Pool Operation

Description:

(1) Task A gets a memory block, leaving one memory block in the memory pool.

(2) Task B also gets a memory block, leaving no memory block in the memory pool.

(3) Task A tries to get another memory block. However, there are no available memory blocks and task A enters the WAIT state.

(4) Task B releases the memory block, which is allocated to task A; task A is released from the WAIT state.

2.7.2 Variable-Size Memory Pool

A task can get a variable-size memory block in a byte unit from the variable-size memory pool.

Figure 2.8 shows how the memory pool works.

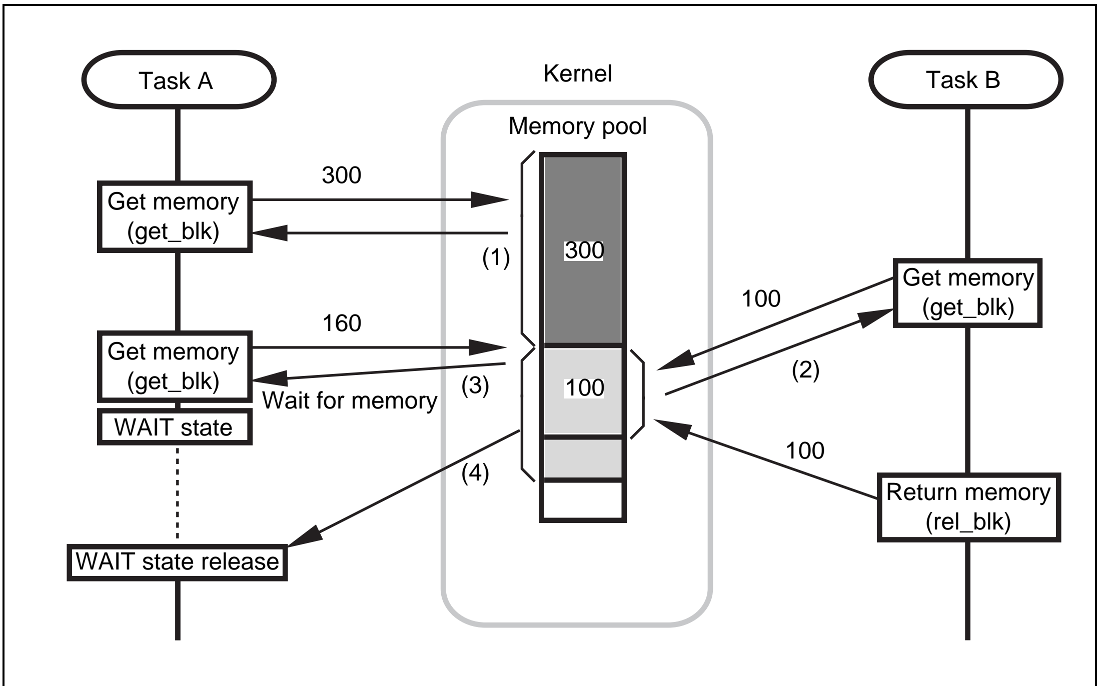


Figure 2.8 Variable-Size Memory Pool Operation

Description:

(1) Task A obtains 300 bytes of memory area, with additional 16 bytes for OS management purposes, thus leaving 184 (= 500 – 300 – 16) bytes of available memory area.

(2) Task B also obtains 100 bytes of memory area, with additional 16 bytes for OS management purposes, thus leaving 68 bytes of available memory area.

(3) Task A tries to obtain 160 bytes of memory area. However, there are only 68 (= 184 – 100 – 16) bytes of available memory area and thus task A enters the WAIT state.

(4) Task B releases 100 bytes, with 16 bytes for OS management purposes, making 184 bytes available. Accordingly, 160 bytes of the memory area are allocated to task A, and task A is released form the WAIT state. Here, 8 (= 68 + 100 + 16 – 160 – 16) bytes of the memory area are left.

2.8 Time Management

2.8.1 Overview

The kernel manages time using a clock of a given frequency generated using a hardware timer. This provides the following functions:

Reference to and Setting of Time: Manages time by counting the pulses of the hardware clock at a certain point specified by the system.

Timer Handler Execution Control: Monitors the cyclic elapsed time of the cyclic handler, and controls execution.

Task Execution Control: Controls execution of tasks using time.

To use the above functions, a timer handler must be created by the user. For details on the timer handler creation, refer to appendix C, Device Driver.

System clock is operated by the system calls listed in table 2.10.

Table 2.10 System Calls for System Clock

System Call Name	Function
set_tim	Sets system clock
get_tim	Refers to system clock

2.8.2 Hardware Timer and System Clock

The time management requires a hardware timer to generate interrupts with a certain cycle time. The kernel counts the interrupts using the hardware timer and manages their timing. The unit of time used in the operating system (system clock value) is cycle time of the hardware timer (tc). The relationship between time in the operating system (OS) and actual time is:

<actual time> = <time in OS (system clock value)> x <cycle time of hardware timer (tc)>

When the hardware timer cycle is 1 ms, a value of 100 in <time in OS> indicates 100 ms (<actual time>).

2.8.3 Setting and Referring to System Clock

A 48-bit signed system clock counter is incremented by one each time the hardware timer interrupt is generated. This enables the time to be calculated up to about 1.4×10^{14} (about 4,000 years when the cycle time of the hardware timer (tc) is 1 ms).

2.8.4 Cyclic Handler

Cycle processing can be performed by using the cyclic handler. The cyclic handler can be initiated when the system is in the task-independent portion; it is initiated at a specific cyclic time interval.

Cyclic handlers are controlled by the system calls listed in table 2.11.

Table 2.11 System Calls for Cyclic Handler Control

System Call Name	Function
act_cyc	Controls the activation of the cyclic handler
ref_cyc	Refers the cyclic handler status

Figure 2.9 shows an overview of cyclic handler processing.

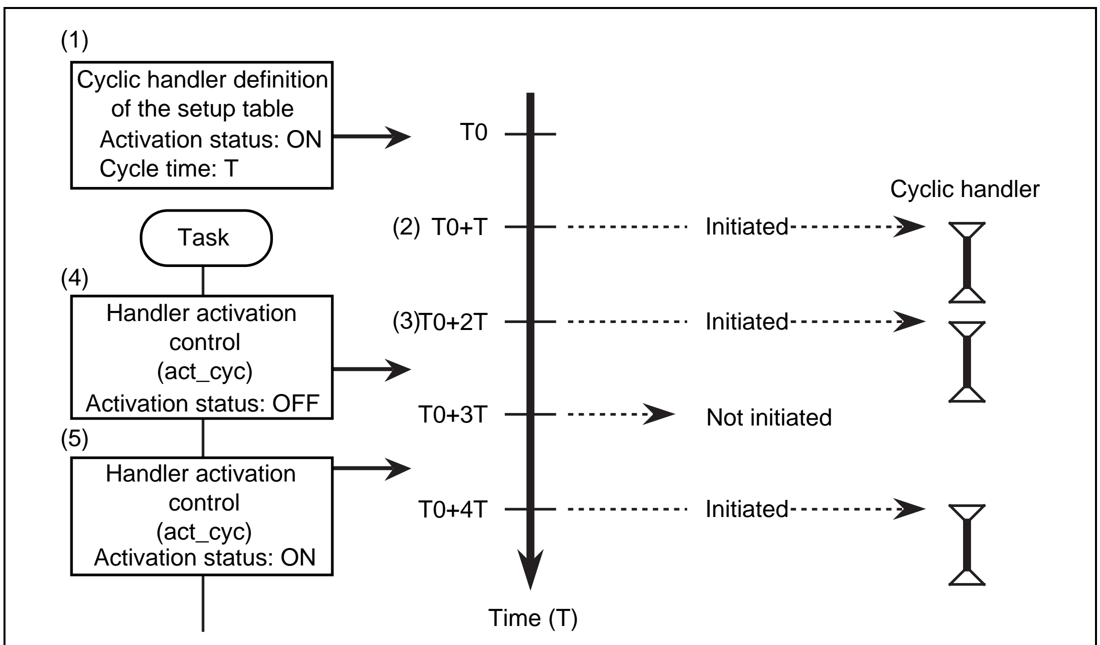


Figure 2.9 Overview of Cyclic Handler Processing

Description:

(1) The cyclic handler activation status is turned on and a cyclic handler with cycle time T is defined in the setup table.

(2) The cyclic handler is initiated after cycle time T has passed.

(3) The cyclic handler is initiated after cycle time T has passed and when time becomes T0 + 2T.

(4) If the activation status is turned off by issuing the act_cyc system call, the cyclic handler will not be initiated even after cycle time T has passed (time is T0 + 3T).

(5) If the activation status is turned on by issuing the act_cyc system call, the cyclic handler will be initiated again when cycle time T has passed (time is T0 + 4T).

2.9 System Management

The kernel version can be acquired by using the system call listed in table 2.12 to manage the system.

Table 2.12 System Call for Kernel Version Acquisition

System Call Name	Function
get_ver	Refers to the version

2.10 System-Call Trace

The system-call trace function stores a history of the system calls issued during program execution in the trace buffer. Basically, information on task issue and task return can be acquired by issuing a system call. The information is called an event.

To use the trace function, the trace function and the trace buffer area must be defined in the setup table at system configuration. When the trace function is selected, all events following the execution of the system initialization handler will be acquired. The trace buffer has a ring-buffer structure and writes new information over old information.

2.11 Trace Buffer Structure

Figure 2.10 shows the trace buffer structure.

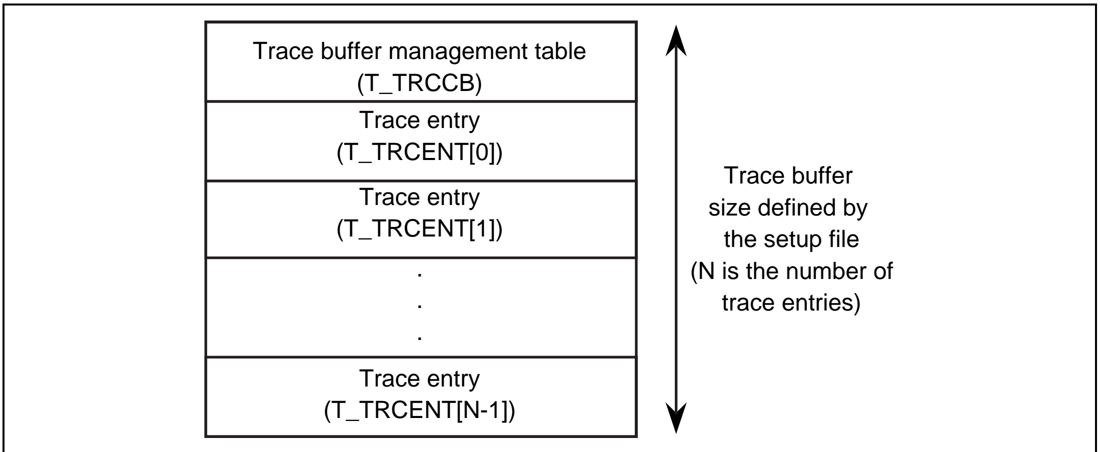
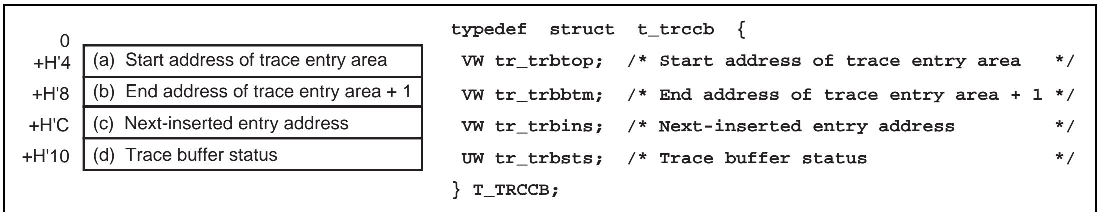


Figure 2.10 Trace Buffer Structure

The trace entry area stores acquired information and has a ring buffer structure. One trace entry area is used for each event.

Trace Buffer Management Table (T_TRCCB): Controls the trace buffer. The kernel uses this area at trace acquisition. Figure 2.11 shows the structure of the trace buffer management table.




```
typedef  struct  t_trccb  {
 VW tr_trbtop;  /* Start address of trace entry area   */
 VW tr_trbbtm;  /* End address of trace entry area + 1 */
 VW tr_trbins;  /* Next-inserted entry address         */
 UW tr_trbsts;  /* Trace buffer status                 */
} T_TRCCB;
```

Figure 2.11 Trace Buffer Management Table Structure

Areas (a) and (b) store the trace information location. These areas are initialized according to the setup table at system initiation.

Area (c) stores the address where the next-event information is to be stored.

Area (d) contains two valid status bits; the other bits are invalid.

Bit 0: Ring buffer flag
- 0: A complete round of writing to the trace buffer has not yet been made
- 1: At least one complete round of writing to the trace buffer has been made

Bit 1: Trace acquisition flag

Set to 1 while the kernel is storing trace information in the next-inserted entry address, which means that the information in area (c) is undefined.

Figure 2.12 shows the trace buffer management process.

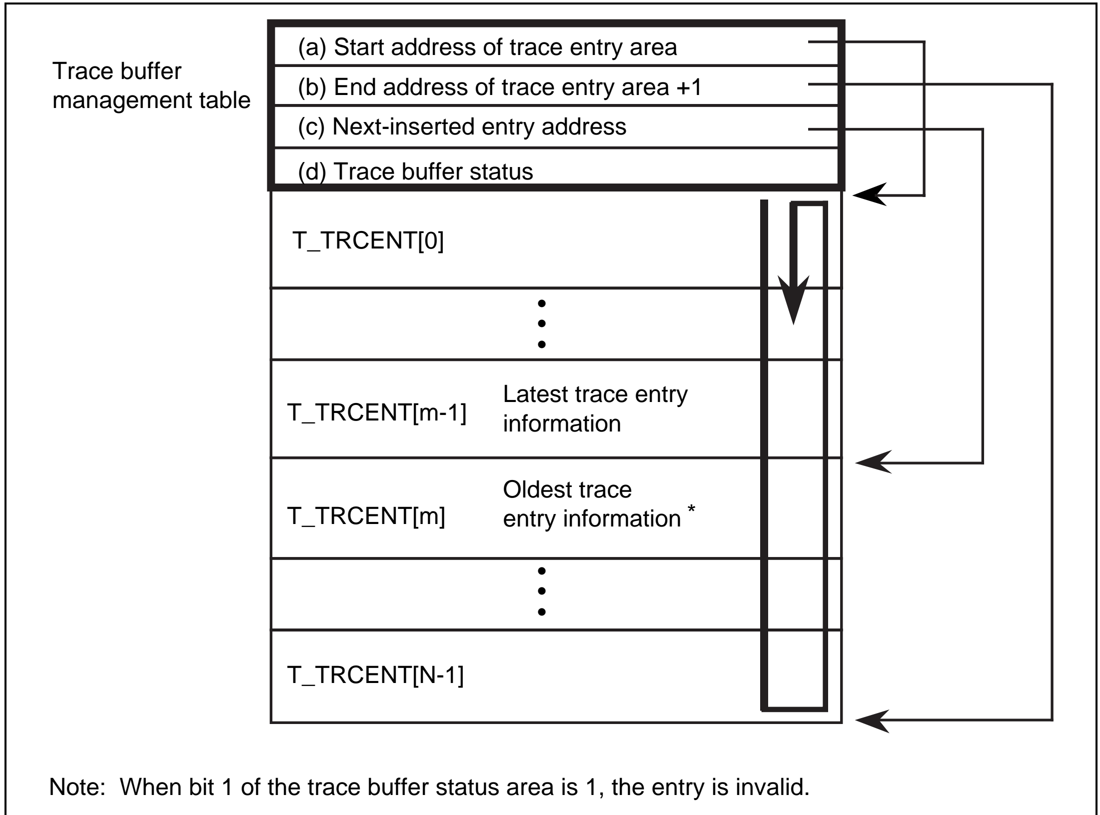


Figure 2.12 Trace Buffer Management Process

Trace Entry (T_TRCENT): One trace entry stores trace information for one event. Figure 2.13 shows the trace entry structure.

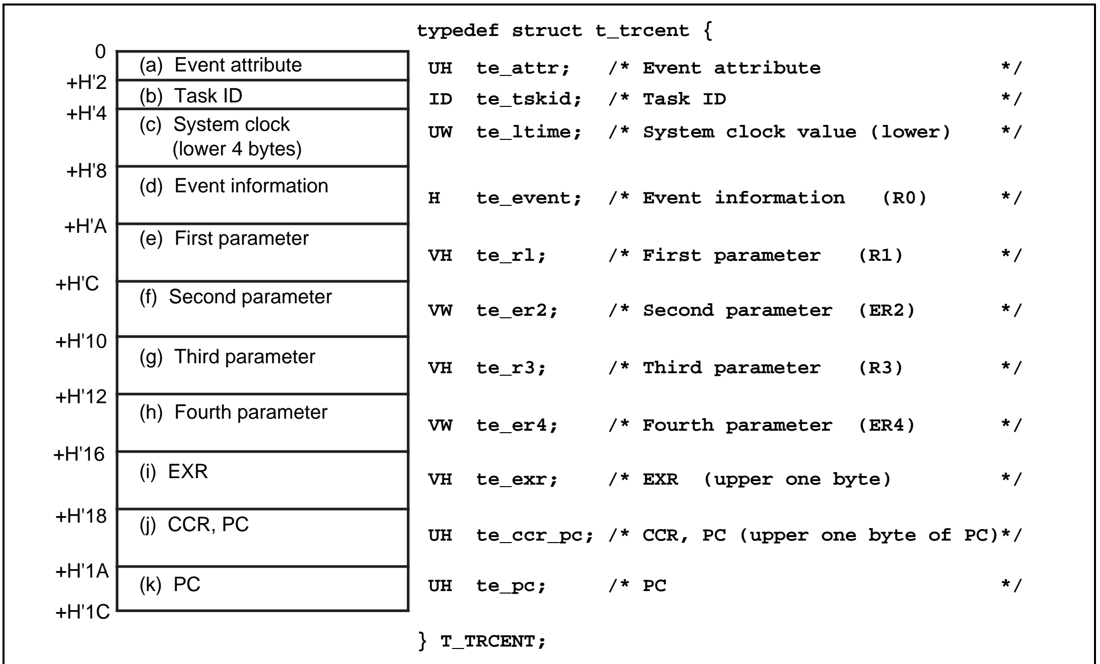


Figure 2.13 Trace Entry Structure

The event attribute indicates the type of trace entry. An event can have one of the following four attributes:

- SVC attribute (TATR_SVC: H'0001)
- RTN attribute (TATR_RTN: H'0002)
- CONT attribute (TATR_CONT: H'0003)
- IDLE attribute (TATR_IDLE: H'0004)

The SVC attribute indicates that the event is the issuing of a system call, so the trace entry stores the information at the time the system call is issued.

The RTN attribute indicates that the event is a return from the kernel to the application, so the trace entry stores the information at a system call return or at a task or handler initiation.

The CONT attribute event is acquired when task execution restarts from the interrupted point according to case 3 below, one of three possible ways to restart execution.

1. When returning to the task that was running before the interrupt as a result of the interrupt handler executing the RTE instruction.

2. When the interrupt handler issues the ret_int system call without issuing a system call (including interrupts from the system timer) that requires task switching, and thus execution returns to the task that was executing before the interrupt.
3. When the system call ret_int is issued after a system call that requires task switching is issued (including interrupts from the system timer), execution returns to a task other than the task that was running before the interrupt, and afterwards returning control to the task that was running before the interrupt.

The CONT-attribute event is acquired when 3 is satisfied. In cases 1 or 2, it will not be acquired.

The IDLE attribute event is acquired when the system enters the idle state.

The meaning of the other trace entry data depends on the event attribute. Table 2.13 shows the possible meanings.

Table 2.13 Trace Entry Data Meanings

	Event Attribute (te_atr)			
te_atr	**TATR_SVC (H'0001)**	**TATR_RTN (H'0002)**	**TATR_CONT (H'0003)**	**TATR_IDLE (H'0004)**
te_tskid	ID of task issuing system call. 0 when issued from task-independent portion.	ID of task to which execution returns from the kernel. 0 when returning to task-independent portion.	ID of task to restart from an interrupt point. Never the task-independent portion (0).	Undefined.
te_ltime	Lower four bytes of the system clock count at event acquisition.			
te_event	Function code of the issued system call.* (the function code of the ret_int system call will not be acquired)	Error code of the system call. However, H'8000 will mean task initiation (R0).	Undefined.	Undefined.
te_r1, te_er2, te_r3, te_er4	System-call parameters (R1, ER2, R3, and ER4 at system-call issue).	System-call return parameters. Data is not defined at task initiation (R1, ER2, R3, ER4 at system-call return)	Undefined.	Undefined.
te_exr	EXR at system-call issue	EXR at application return	Undefined.	Undefined.
te_ccr_pc	CCR at system-call issue	CCR at application return	Undefined.	Undefined.
te_pc	System-call issue address	Application return address. When a task is initiated, the task start address will be returned. In other cases, the issue address of the previously issued system call will be returned.	Undefined.	Undefined.

Note: For details on the system-call function code, refer to appendix E, System-Call Function Codes.

2.12 Trace Acquisition Data Analysis Example

An example of acquired trace data is shown in table 2.14.

Table 2.14 Trace Acquisition Data Example

No.		te_attr	te_tskid	te_ltime	te_event	te_r1	te_pc	te_ccr	te_exr
–12	Old ↑	H'0001 SVC	H'0005 tskid = 5	H'00001234	H'ffe9 sta_tsk	H'0003 Starts ID = 3	H'003018	H'00	H'00
–11		H'0002 RTN	H'0003 tskid = 3	H'00001234	H'8000 Task initiation	H'xxxx	H'003800	H'00	H'00
–10		H'0001 SVC	H'0003 tskid = 3	H'00001234	H'ffda slp_tsk	H'xxxx	H'003810	H'00	H'00
–9		H'0002 RTN	H'0005 tskid = 5	H'00001234	H'0000 E_OK	H'0003	H'003018	H'00	H'00
–8		H'0001 SVC	H'0000 Non-task	H'00001234	H'ff87 iwup_tsk	H'0003 Wakes up ID = 3	H'007340	H'00 Control level = 0	H'05 Interrupt level = 5
–7		H'0002 RTN	H'0000 Non-task	H'00001234	H'0000 E_OK	H'0003	H'007340	H'00 Control level = 0	H'05 Interrupt level = 5
–6		H'0002 RTN	H'0003 tskid = 3	H'00001234	H'0000 E_OK	H'xxxx	H'003810	H'00	H'00
–5		H'0001 SVC	H'0003 tskid = 3	H'00001234	H'ffe1 rel_wai	H'0005 Releases ID = 5 from wait state	H'003840	H'00	H'00
–4		H'0002 RTN	H'0003 tskid = 3	H'00001234	H'ffc1 E_OBJ	H'0005	H'003840	H'00	H'00
–3		H'0001 SVC	H'0003 tskid = 3	H'00001234	H'ffeb ext_tsk	Undefined data	Undefined data		
–2		H'0003 CONT	H'0005 tskid = 5	H'00001234	Undefined data	Undefined data	Undefined data		
–1	↓	H'0001 SVC	H'0005 tskid = 5	H'00001234	H'ffeb ext_tsk	Undefined data	Undefined data		
0	New	H'0004 IDLE	Undefined	H'00001234	Undefined data	Undefined data	Undefined data		

Notes: 1. te_er2, te_r3, and te_er4 are omitted to simplify description.
2. In each event row, the upper line shows the traced data, and the lower line briefly describes the data.
3. Numbers are only for description. They are not acquired as trace data.

1. Event No. –12

   This is the SVC attribute because te_attr is H'0001. te_tskid = 5 indicates that task 5 has issued a system call. The system call is sta_tsk because te_event, which shows the system call function code, is H'FFE9. System call sta_tsk has initiated task 3 because te_par1, which is system call parameter tskid, is te_r1=H'0003. The address of the instruction issuing the sta_tsk system call is H'3016 (= 3018 – 2) because te_pc is H'003018.

2. Event No. –11

   te_attr = H'0002 (RTN attribute) and te_tskid = 3 indicate that control has moved to task 3. te_event = H'8000 indicates that task 3 started at this time. The task start address is te_pc = H'003800.

   Due to its relation with event –12, task 5 issued a sta_tsk system call to switch control from task 5 to task 3.

3. Event No. –10

   te_attr = H'0001 (SVC attribute), te_tskid = 3, and te_event = H'FFDA indicate that task 3 has issued slp_tsk.

4. Event No. –9

   te_attr = H'0002 (RTN attribute) and te_tskid = 5 indicate that control has moved to task 5. Due to its relation with event –10, system call slp_tsk issued by task 3 has switched control from task 3 to task 5 here. Task 5 has not been executed (no events have been acquired) since the sta_tsk system call was issued at event –12; therefore, te_event is the error code for the event –12 sta_tsk system call.

5. Event No. –8

   te_attr = H'0001 (SVC attribute), te_tskid = 0, te_event H'FF87, and te_r1 = 3 indicate that iwup_tsk (tskid = 3) has been issued from a task-independent portion. A task-independent portion may be an interrupt handler, extended SVC handler, or system initialization handler. In this case, te_ccr = H'00 and te_exr = H'05 indicates that the task-independent portion is the interrupt handler having interrupt level 5 with a priority level of 0. In this case, an interrupt occurred between events –9 and –8.

6. Event No. –7

   te_attr = H'0002 (RTN attribute) and te_tskid = 0 indicate that the information is on a return from the system call issued from a task-independent portion. iwup_tsk in event –8 is the previous system call issued by a task-independent portion; therefore te_event is the error code for this iwup_tsk system call.

7. Event No. –6

   te_attr = H'0002 (RTN attribute) and te_tskid = 3 indicate that control has moved to task 3. The previous event, –7, is for a task-independent portion; therefore, the ret_int system call must have been issued from an interrupt handler between events –7 and –6. As a result, task 3 was given control and this event was acquired. Task 3, therefore, has a higher priority than task 5. Task 3 has not been executed since the event –10 slp_tsk system call was issued; therefore, te_event is the error code for this slp_tsk system call.

8. Event No –5

   te_attr = H'0001 (SVC attribute), te_tskid = 3, te_event = H'FFE1, and te_r1 = H'5 indicate that task 3 has issued rel_wai (tskid = 5).

9. Event No. –4

   te_attr = H'0002 (RTN attribute), te_tskid = 3, and te_event = H'FFC1 indicate that the system call rel_wai (event –5) issued by task 3 has resulted in an error (error code: E_OBJ).

10. Event No. –3

    te_attr = H'0001 (SVC attribute), te_tskid = 3, and te_event = H'FFEB indicate that task 3 has issued the ext_tsk system call.

11. Event No. –2

    te_attr = H'0003 (CONT attribute) and te_tskid = 5 indicate that task 5 restarted from the interrupt point. Because task 3, which was being executed, has issued the ext_tsk system call (event –3), task 5 was given a control. Checking previous trace data shows that data on task 5 is not found from event –9 until this event. Therefore, an interrupt has occurred between events –9 and –8, and this interrupt suspended task 5 execution.

12. Event No. –1

    te_attr = H'0001 (SVC attribute), te_tskid = 5, and te_event = H'FFEB indicate that task 5 has issued the system call ext_sk.

13. Event No. 0

    te_attr = H'0004 (IDLE attribute) indicates that the system has entered idling state.

The program flow for the trace data in table 2.14 is shown in figure 2.14.

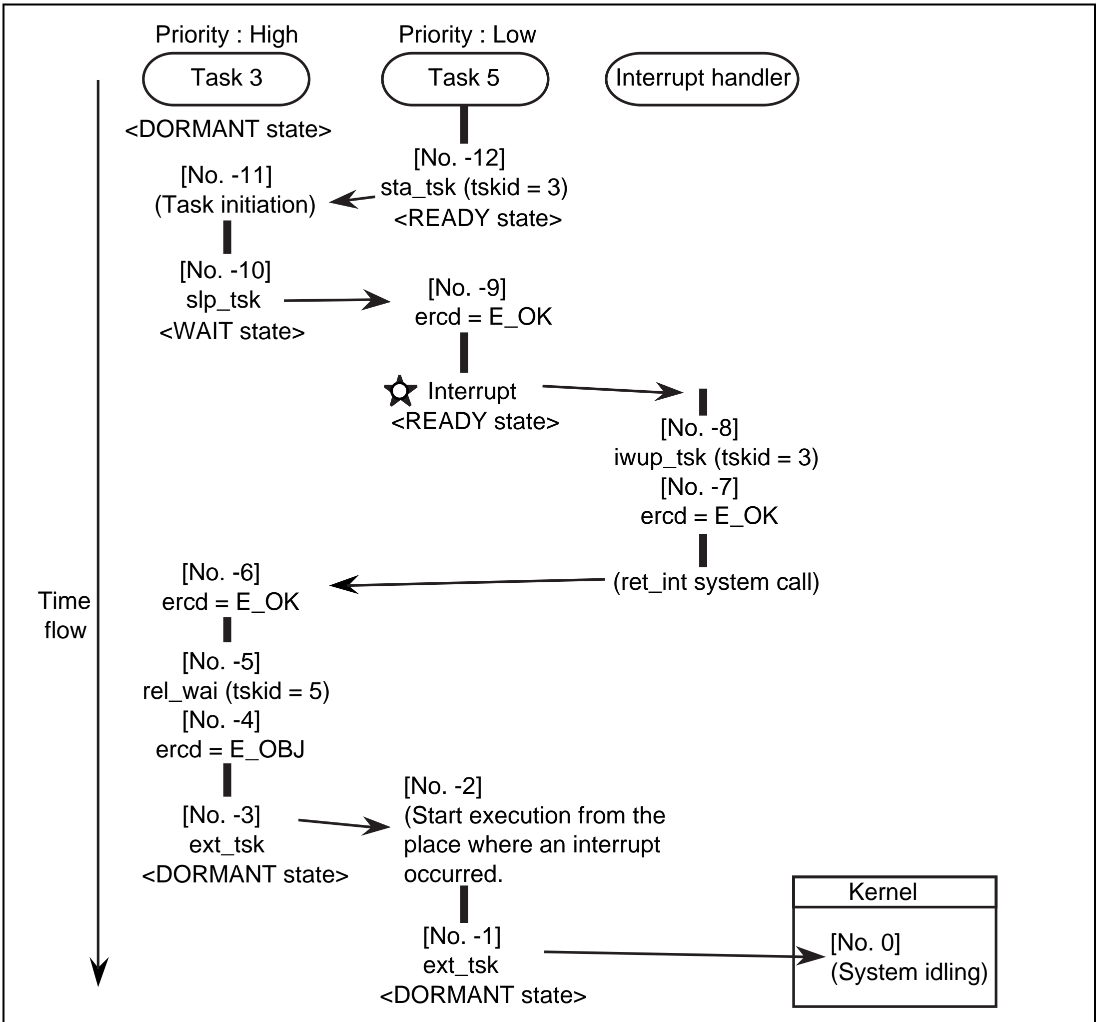


Figure 2.14 Example of Trace Analysis Results

2.13 Trace-Function Definition

For details on the trace-function definition, refer to section 6.2.6, Defining Trace Functions.

2.14 Notes on Trace Function

1. Kernel performance degradation

   When the trace function is used, trace acquisition processing increases the system-call processing time because the trace acquisition process is added to system-call processing. It also increases the interrupt-inhibited time for the kernel. In some systems, these increases may cause timing problems.

   These problems also depend on the memory size (location) that trace the buffer area is allocated to.

2. Writing to the trace buffer

   The trace buffer must not be written to. In particular, if the data in the trace buffer management table is changed, correct system operation is not guaranteed.

3. Trace information concerning the RTN attribute

   The E2 register value is not defined in the trace information in the normal mode for the RTN attribute.

4. Interrupt control mode

   The EXR register value is not defined in the trace information when interrupt control mode 0 or 1 is selected.

   The CCR register value is not defined in the trace information when interrupt control mode 2 is selected.

Section 3 System Calls

3.1 Overview

System calls are classified as shown in table 3.1.

Table 3.1 System Call Classification

Classification	Description
Task management function	Initiates and terminates tasks
Task synchronization function	Suspends and resumes task execution and task event flag
Synchronization and communication function	Manages event flags, semaphores, and mailboxes
Interrupt management function	Returns from the interrupt handler, and changes and references the interrupt mask
Memory pool management function	Allocates memory dynamically
Time management function	Sets and references the system clock, and defines the timer handler
System management function	Refers to kernel version identifiers

Some system calls dedicated to task-independent portion have "i's" added at the beginning of the system call name, while others do not. This means that some system call names change when issued from a task portion and a task-independent portion.

- System calls names that change: sta_tsk and ista_tsk, rot_rdq and irot_rdq, wup-tsk and iwup_tsk, and others
- System calls names that do not change: get_tid, ref_tsk, can_wup, and others

3.2 System Call Interface

System calls can be issued from programs written in C or assembly language. This section describes how to issue system calls. For details, see the section 3.3.

3.2.1 C-Language Interface

The kernel provides a C-language interface library so that system calls can be issued from tasks and handlers written in C language.
The C-language interface library consists of library files and C language header files. Library files are provided for the 2600CPU normal mode and advanced mode and for the 2000CPU normal mode and advanced mode.

To issue a system call from a program written in C language, include a C language header file in the source program and link the C-language interface library to the compiled source program (object file) during system configuration. Library files are provided for the 2600 CPU normal mode and advanced mode, and the 2000 CPU normal mode and advanced mode. When issuing a system call from a program written in C language, include C-language header files in the source programs, and link C-language interface libraries at system configuration.

System Call Issue Format: The kernel has the following basic format for system calls written in C language.

ercd = <name> ([[<return parameter address>...],<parameter>...]);
ercd = <name> (void);
void <name> (void);

ercd:	Error code (signed 16-bit integer) acquired as return value of a system call
<name>:	System call name
<return parameter address>:	Address for return parameters (pointer)
<parameter>:	Parameters
void:	Function which cannot receive a return value or function without a parameter

Parameter Name Abbreviation: The following prefixes or suffixes are used for parameters.

Table 3.2 Parameter Prefixes and Suffixes

Prefix and Suffix	Parameter
t_~	Structure
E_~	Error code
p_~	Pointer
pk_~	Parameter packet address
ppk_~	Parameter packet address pointer
~id	ID
~cd	Code
i~	Initial value
~sz	Size
~cnt	Count

Type and Size of Parameter Data: The following list shows the type and size of parameter data used in the kernel. These are defined in the kernel C language standard header file.

```
typedef char            B;          /* signed 8-bit integer             */
typedef short           H;          /* signed 16-bit integer            */
typedef long            W;          /* signed 32-bit integer            */
typedef unsigned char   UB;         /* unsigned 8-bit integer           */
typedef unsigned short  UH;         /* unsigned 16-bit integer          */
typedef unsigned long   UW;         /* unsigned 32-bit integer          */
typedef char            VB;         /* variable data type (8 bits)      */
typedef short           VH;         /* variable data type (16 bits)     */
typedef long            VW;         /* variable data type (32 bits)     */
typedef void            *VP;        /* pointer to variable data type    */
typedef void            (*FP)();    /* program start address (general) */
typedef H               INT;        /* signed 16-bit integer            */
typedef UH              UINT;       /* unsigned 16-bit integer          */
typedef INT             BOOL;       /* Boolean value FALSE(0) or        */
                                    /* TRUE(1)                          */
typedef int             FN;         /* function code                    */
typedef UH              ID;         /* object ID number (???id)         */
typedef ID              BOOL_ID;    /* Boolean value or ID number       */
typedef H               HNO;        /* handler number                   */
typedef H               ER;         /* error code                       */
typedef H               PRI;        /* task priority                    */
typedef W               TMO;        /* timeout                          */
typedef TMO             CYCTIME;    /* cyclic time initiation interval */
```

C-Language Interface Description Example: The following shows a C-language interface parameter description, using a sta_tsk system call as an example.

```
#include "hi2000.h"
void        task(INT stacd)
{
ER          ercd;
ID          tskid;
INT         stacd;
            /* ...                                  */
            ercd = sta_tsk(tskid,stacd);
            /* ...                                  */
}
```

3.2.2 Assembler Interface

System Call Issue Format: The kernel has the following basic format for system calls in the assembly language program. After parameters have been set in each register, a JSR instruction is executed. An example of system call sta_tsk is shown below.

```
MOV.W    #TSKID,R1                          (a)
MOV.W    #STACD,R2                          (b)
JSR      @sta_tsk                           (c)
```

(a) System call parameter (task ID) is set in the registers defined by each system call.

(b) System call parameter (initiation code) is set in the registers defined by each system call.

(c) The JSR instruction is executed in the format defined for each system call. Some system calls use the JMP instruction.

At system call termination, an error code is returned to register R0. Registers other than R0 and parameter registers maintain the value before the system call was issued.

Constants Used in Parameters: The kernel provides an assembly language header file. Various constants are defined in the assembly language header file.

3.2.3 Error Codes

Except for a few system calls, error codes are returned as system call execution results. Error codes are set in register R0 as an ER type (signed 16-bit integer). The results of system call execution are not reflected in each flag of the CCR register.

For the system calls described in this section, error codes that may be generated are described.

The kernel provides two types of library kernels: one has a parameter check function and the other does not. If the latter type is used, the kernel omits the static error detection for the system call parameter, reducing the system call processing time; therefore, when there is an error in the system call parameter, correct system operation cannot be guaranteed.

Usually, a kernel library with a parameter check function is used for debugging. Then, after debugging has ended, the kernel library without a parameter check function is used.

3.3 System Calls

In this section, system calls are described in details as shown below.

Section	Brief function description (System call name)	[System status enabling system call issuing]

C Language Interface:

System call issuing format

Assembler Interface:

System call issuing format

Parameters:

Type	Parameter name	Register	Meaning of parameter
⋮	⋮	⋮	⋮

Return Parameters:

Type	Parameter name	Register	Meaning of parameter
⋮	⋮	⋮	⋮

Packet Structure:

Error code:

Mnemonic	Error code value	[Type]	Meaning of error code
⋮	⋮	⋮	⋮

Description:

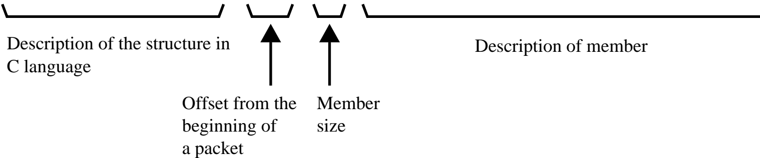
- System status enabling system call issuing:
  The following mnemonics show the system status in which a system call can be issued.
  T: Task-execution state
  D: Dispatch-disabled state
  L: CPU-locked state
  I: Task-independent portion

- Register (parameter/return parameter):
  ERx/Rx: The register size differs between the advanced and normal modes
  Rx: The register size is the same in the advanced and normal modes

- System call name:
  If (System call) is written in the parameter, return parameter, or error code, it denotes the target system call.

- Packet Structure
Packets are described as below when used by a system call.

```
typedef    struct  t_rsem{
           VP  exinf;          0/0     4/2    Extended information
           BOOL_ID  wtsk;    +4/+2     2/2    Wait task ID
           UINT  semcnt;     +6/+4     2/2    Current semaphore count
}T_RSEM;
```



Offset from the beginning of a packet and member size
x/xx: The offset from the beginning of a packet and the member size differs between advanced mode or normal mode.
x: The offset from the beginning of a packet and the member size is the same in advanced mode or normal mode.

- Error Code Type
  [k] indicates an error that is detected regardless of the parameter check function.
  [p] indicates an error that is detected only when the parameter check function is incorporated.

Figure 3.1 System Call Description Form

3.4 Task Management

Task-Management System Calls: Tasks are managed by the system calls listed in table 3.3.

Table 3.3 System Calls for Task Management

System Call	Description	System State T/D/L/I
sta_tsk	Starts task	T/D/L
ista_tsk	Starts task (task-independent potion)	D/I
ext_tsk	Terminates current task	T/D/L
ter_tsk	Forcibly terminates a task	T/D/L
chg_pri	Changes task priority	T/D/L
rot_rdq	Rotates task ready queue	T/D/L
irot_rdq	Rotates task ready queue (task-independent portion)	D/I
rel_wai	Releases the task WAIT state	T/D/L
get_tid	Refers current task ID	T/D/L
ref_tsk	Refers task state	T/D/L/I
dis_dsp	Disables dispatch	T/D
ena_dsp	Enables dispatch	T/D

Task Management Specifications: Task-management specifications are listed in table 3.4.

Table 3.4 Task-Management Specifications

Item	Description
Maximum number of tasks that can be defined	225
Task ID	1 to 255 (including undefined tasks)
Task priority	1 to 31
Task stack	Includes shared-stack function
Ready queue	First-come first-service (FCFS)
Shared stack queue (when the shared-stack function is used)	First-in first-out (FIFO)

Task-Execution Waiting/Suspension and Release: Table 3.5 lists the causes of task-execution waiting/suspension and release.

Table 3.5 Causes of Task-Execution Waiting/Suspension and Release

Cause of Waiting/Suspension		Time of Release
When an interrupt is generated		When an interrupt handler completes execution
When a shared stack is being occupied	sta_tsk or ista_tsk system call	When the shared stack is released

3.4.1 Start Task (sta_tsk) [T/D/L]
Start Task (ista_tsk) [D/I]

C-Language Interface:

```
ER ercd = sta_tsk (ID tskid, INT stacd);
ER ercd = ista_tsk (ID tskid, INT stacd);
```

Assembler Interface:

```
JSR        @sta_tsk
JSR        @ista_tsk
```

Parameters:

ID	tskid	R1	Task ID
INT	stacd	R2	Task initiation code

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid ≤ 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid does not exist)
E_OBJ	H'ffc1 (–H'3f)	[k]	Object state is invalid (Task indicated by tskid is not in DORMANT state)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call ista_tsk while tasks were being executed or a task-independent portion issued system call sta_tsk)
		[k]	(System call ista_tsk was issued from a task portion while the CPU was being locked)

Description:

These system calls initiate the task indicated by the parameter tskid. The initiated task makes a transition from the DORMANT state to the READY state. The task initiation code indicated by the parameter stacd will be passed to the initiated task. Parameter stacd must be passed to the R0 register when the task is written in assembly language, and must be passed to the first argument when written in C language. Initiation requests are not queued. Parameter tskid specifies the ID of the task to be initiated. The current task cannot be specified by the parameter tskid.

If the shared stack is not used by another task, the task to be initiated uses the shared stack and shifts to the READY state. If the shared stack is already used by another task, the task indicated by tskid shifts to the WAIT state and is placed in the shared-stack-wait queue since the stack area cannot be used. If this system call is issued to a task in the shared-stack-wait state, error code E_OBJ is returned.

3.4.2 Exit Task (ext_tsk) [T/D/L]

C-Language Interface:

```
void ext_tsk (void);
```

Assembler Interface:

```
JMP          @ext_tsk
```

Parameters:

None

Return Parameter:

None

Error Codes:

Normal termination:	[k]	Does not return to the task that issued this system call.
Abnormal termination:	[p]	If a task-independent portion has issued this system call, control is passed to the system termination routine.

Description:

The system call ext_tsk terminates the current task.

After the execution of this system call, the current task makes a transition from the RUN state to the DORMANT state. The system call ext_tsk cannot automatically release the resources acquired by the semaphore or the memory blocks acquired before the task is terminated. Therefore, the user must issue system calls to release resources and memory blocks before issuing the system call ext_tsk.

Therefore, if the current task shares the stack with other tasks, the task at the head of the stack wait queue is removed and is placed in the READY state.

A task portion can issue the system call ext_tsk while task dispatch is being disabled or while the CPU is being locked. If issued, the kernel enables the execution of other tasks.

3.4.3 Terminate Task (ter_tsk) [T/D/L]

C-Language Interface:

```
ER ercd = ter_tsk (ID tskid);
```

Assembler Interface:

```
JSR          @ter_tsk
```

Parameters:

ID	tskid	R1	Task ID

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid < 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid does not exist)
E_OBJ	H'ffc1 (–H'3f)	[p] [k]	Object state is invalid (Task is in DORMANT state) (Current task is specified)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call ter_tsk)

Description:

The system call ter_tsk forces a task specified by tskid to terminate. The terminated task enters DORMANT state.

The parameter tskid specifies the ID of the task to be terminated.

The system call ter_tsk cannot release resources acquired by the semaphore or memory blocks acquired. Therefore, the user must issue system calls to release resources and memory blocks before issuing the system call ter_tsk.

If the current task shares the stack with other tasks, the task at the head of the stack wait queue is removed and placed in the READY state.

3.4.4 Change Task Priority (chg_pri) [T/D/L]

C-Language Interface:

```
ER ercd = chg_pri (ID tskid, PRI tskpri);
```

Assembler Interface:

```
JSR         @chg_pri
```

Parameters:

ID	tskid	R1	Task ID
PRI	tskpri	R2	Task priority (0 to maximum task priority)

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Parameter error (tskpri < 0, tskpri > Maximum task priority)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid < 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task specified by tskid is undefined)
E_OBJ	H'ffc1 (–H'3f)	[k]	Object status is incorrect (Task is in DORMANT state)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call chg_pri)

Description:

The system call chg_pri changes the priority of the task specified by the parameter tskid to the value specified by the parameter tskpri. The current task can also be specified by specifying tskid = TSK_SELF (0).

The parameter tskpri specifies the task priority ranging from 0 to the Maximum task priority. The task with the smallest value has the highest priority.

Specifying tskpri = TPRI_INI (0) returns the task priority to the initial priority that was specified at task definition.

A priority changed by this system call is valid until the task is terminated or until this system call is issued again. If a task enters the DORMANT state, its previous task priority before termination becomes invalid and it returns to the initial task priority specified at task definition.

3.4.5 Rotate Ready Queue (rot_rdq) [T/D/L]
Rotate Ready Queue (irot_rdq) [D/I]

C-Language Interface:

```
ER ercd = rot_rdq (PRI tskpri);

ER ercd = irot_rdq (PRI tskpri);
```

Assembler Interface:

```
JSR    @rot_rdq

JSR    @irot_rdq
```

Parameters:

PRI	tskpri	R2	Task priority

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Parameter error (tskpri < 0, tskpri > Maximum task priority)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call irot_rdq while tasks were being executed or a task-independent portion issued system call rot_rdq)
		[k]	(A task portion issued system call irot_rdq while the CPU was being locked)

Description:

These system calls rotate the ready queue of the task priority indicated by the parameter tskpri. In other words, the task at the head of the task priority ready queue is sent to the end, enabling the second task in the ready queue to be executed.

The parameter tskpri specifies the task priority ranging from 0 to the Maximum task priority.

Specifying tskpri = TPRI_RUN (0) rotates the ready queue (the ready queue with the highest priority) including the task being executed. However, while task dispatch is disabled, the task in the execution (RUN) state may not have the highest priority.

If tskpri = TPRI_RUN (0) or the priority of the current task is specified, the current task is sent to the end of the ready queue and it releases control.

If the specified ready queue is empty or if there are no tasks in the RUN state, these system calls have no effect; and the task terminates normally.

3.4.6 Release Wait (rel_wai) [T/D/L]

C-Language Interface:

ER ercd = rel_wai (ID tskid);

Assembler Interface:

JSR @rel_wai

Parameters:

ID	tskid	R1	Task ID

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid ≤ 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid does not exist)
E_OBJ	H'ffc1 (–H'3f)	[k]	An object status is invalid (Task indicated by tskid is the current task or is a task not in WAIT state)
E_CTX	H'ffbb (–H'45)	[p]	(A task-independent portion issued system call rel_wai)

Description:

If the task specified by tskid is in the WAIT state, the system call rel_wai releases the task from the WAIT state. Note that the SUSPEND or shared-stack WAIT state is not considered as a WAIT state here. The parameter tskid specifies the task ID to release from the WAIT state. To the task specified by tskid, that is, the task that has been released from the WAIT state by the system call rel_wai, error code E_RLWAI is returned.

Note that the system call rel_wai cannot release a task from the SUSPEND state. To release the task from the SUSPEND state, rsm_tsk must be issued. If this system call is issued to a task in the WAIT-SUSPEND state, the task enters the SUSPEND state, which can then be released by the system call rsm_tsk. In this case, error code E_RLWAI is returned.

Note that the system call rel_wai cannot release a task from the shared-stack WAIT state.

3.4.7 Get Task Identifier (get_tid) [T/D/L]

C-Language Interface:

```
ER ercd = get_tid (ID *p_tskid);
```

Assembler Interface:

```
JSR         @get_tid
```

Parameters:

ID	*p_tskid	---	Start address of the area where the task ID is to be returned (C-language interface)

Return Parameters:

ID	*p_tskid	---	Start address of the area where the task ID was stored (C-language interface)
---	tskid	R1	Task ID (Assembler interface)
ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call get_tid)

Description:

The system call get_tid gets the current task ID.

3.4.8 Refer Task State (ref_tsk) [T/D/L/I]

C-Language Interface:

ER ercd = ref_tsk (T_RTSK *pk_rtsk, ID tskid);

Assembler Interface:

JSR @ref_tsk

Parameters:

ID	tskid	R1	Task ID
T_RTSK	*pk_rtsk	ER2/R2	Start address of the packet where the task state is to be returned

Return Parameters:

T_RTSK	*pk_rtsk	ER2/R2	Start address of the packet where the task state is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef struct    t_rtsk {
                  VP exinf;       0/0      4/2   Extended information
                  PRI tskpri;     +4/+2    2/2   Current priority of the task
                  UINT tskstat;   +6/+4    2/2   Task state
                  UINT tskwait;   +8/+6    2/2   Wait cause
                  ID wid;         +10/+8   2/2   Wait object ID
                  H wupcnt;       +12/+10  2/2   Wakeup request count
                  FP task;        +14/+12  4/2   Task start address
                  PRI itskpri;    +18/+14  2/2   Priority at task initiation
                  }T_RTSK;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_rtsk is 0 or an odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid< 0, tskid > Number of tasks defined) (Zero can be specified for parameter tskid only in the task-independent portion)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid does not exist)

Description:

The system call ref_tsk reads the state of the task indicated by the parameter tskid and returns it to the area specified by the parameter pk_rtsk. Note that a 20-byte (advanced mode) or 16-byte (normal mode) RAM area must be defined for the area specified by pk_rtsk.

The following information is returned to the area specified by pk_rtsk:

Note that data with an asterisk * is invalid when the task in the DORMANT state.

exinf Indicates the extended information specified at task definition.

tskpri Indicates the current priority of the task.

tskstat Indicates the current task state. The following values are returned.

tskstat	Code	Description
TTS_RUN	H'0001	RUN state
TTS_RDY	H'0002	READY state
TTS_WAI	H'0004	WAIT state
TTS_SUS	H'0008	SUSPEND state
TTS_WAS	H'000c	WAIT-SUSPEND state
TTS_DMT	H'0010	DORMANT state
TTS_STK	H'4000	Shared stack WAIT state
TTS_STS	H'4008	Shared stack WAIT-SUSPEND state

tskwait* Indicates the causes for shifting the task to WAIT state.

Valid when TTS_WAI or TTS_WAS is returned to tskstat and the following values are returned.

tskwait	Code	Description
TTW_SLP	H'0001	Shifted to WAIT state by slp_tsk or tslp_tsk
TTW_FLG	H'0010	Shifted to WAIT state by wai_flg or twai_flg
TTW_SEM	H'0020	Shifted to WAIT state by wai_sem or twai_sem
TTW_MBX	H'0040	Shifted to WAIT state by rcv_msg or trcv_msg
TTW_MPL	H'1000	Shifted to WAIT state by get_blk or tget_blk
TTW_MPF	H'2000	Shifted to WAIT state by get_blf or tget_blf

wid*	Valid when TTS_WAI or TTS_WAS is returned to tskstat and the waiting target object ID is returned
wupcnt*	The current wakeup request count is returned
task	The task start address is returned.
itskpri	The priority at task initiation (initial priority) is returned.

If tskid = TSK_SELF (0) is indicated, the current task will be specified; however, the system call ref_tsk cannot return the current task ID. To acquire the current task ID, issue system call get_tid.

3.4.9 Disable Dispatch (dis_dsp) [T/D]

C-Language Interface:

ER ercd = dis_dsp (void);

Assembler Interface:

JSR @dis_dsp

Parameters:

None

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call dis_dsp)
		[k]	(A task portion issued system call dis_dsp while the CPU was being locked)

Description:

The system call dis_dsp disables task dispatch during task portion execution. In other words, the state of task portion execution changes from task execution to a state where task dispatch becomes disabled. To return to the task execution, system call ena_dsp must be issued.

The following describes the features when task dispatch is disabled.

1. Task dispatch (scheduling) is delayed until the system returns to the task-execution state. Therefore, no task other than the current task can enter the RUN state.

2. Interrupts can be accepted.

3. System calls to shift a task to the WAIT state cannot be issued. If such system call is issued, an error code is returned.

If the task is terminated by system call ext_tsk when task dispatch is disabled during task portion execution, tasks will be dispatched again enabling the execution of other tasks.

The issue of system call unl_cpu while task dispatch is being disabled also enables task dispatch and enables the execution of other tasks.

The task terminates normally when the system call dis_dsp is issued while the task dispatch is disabled; however, this system call will not be queued.

3.4.10 Enable Dispatch (ena_dsp) [T/D]

C-Language Interface:

ER ercd = ena_dsp (void);

Assembler Interface:

```
JSR         @ena_dsp
```

Parameters:

None

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call ena_dsp)
		[k]	(A task portion issued system call ena_dsp while the CPU was being locked)

Description:

The system call ena_dsp enables task dispatch. Issuing this system call while task dispatch is disabled will enable task dispatch and tasks will be executed. Task dispatch (scheduling) is then performed.

The task terminates normally when the system call ena_dsp is issued while tasks are being executed; however, this system call will not be queued.

3.5 Task Synchronization

Task Synchronization System Calls: The system calls for task synchronization are listed in table 3.6.

Table 3.6 Task Synchronization System Calls

System Call	Description	System State T/D/L/I
sus_tsk	Shifts task to SUSPEND state	T/D/L
rsm_tsk	Resumes the execution of a task in SUSPEND state	T/D/L
slp_tsk	Shifts current task to WAIT state	T
tslp_tsk	Shifts current task to WAIT state (with timeout function)	T
wup_tsk	Wakes up task	T/D/L
iwup_tsk	Wakes up task (dedicated to task-independent portion)	D/I
can_wup	Cancels wake-up task	T/D/L

Task Synchronization Specifications: The task synchronization specifications are listed in table 3.7.

Table 3.7 Task Synchronization Specifications

Item	Description
Maximum number of task wake-up request count	255
Task suspend request	No queuing

Task Waiting/Suspension and Release: Table 3.8 lists the causes of task-execution waiting/suspension and release.

Table 3.8 Causes of Task-Execution Waiting/Suspension and Release

Cause of Waiting/Suspension		Time of Release
When the current task enters the WAIT state	slp_tsk or tslp_tsk system call	(1) When system call wup_tsk is issued (2) When the specified timeout period (tmout) has passed (tslp_tsk) (3) When system call rel_wai is issued
When forcibly suspended by another task	sus_tsk system call	When system call rsm_tsk is issued

3.5.1 Suspend Task (sus_tsk) [T/D/L]

C-Language Interface:

```
ER ercd = sus_tsk (ID tskid);
```

Assembler Interface:

```
JSR          @sus_tsk
```

Parameters:

ID	tskid	R1	Task ID

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid ≤ 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid does not exist)
E_OBJ	H'ffc1 (–H'3f)	[p]	Object status is incorrect (The current task is specified)
		[k]	(Task is in DORMANT state)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call sus_tsk)
E_QOVR	H'ffb7 (–H'49)	[k]	Queuing overflow (The task is already in the SUSPEND state)

Description:

The system call sus_tsk suspends execution of the task specified by tskid and shifts the task to the SUSPEND state. The SUSPEND state is released by issuing system call rsm_tsk. If the task specified by parameter tskid is already in the WAIT state, it enters the WAIT-SUSPEND state.

A task enters the SUSPEND state when system call sus_tsk is issued from another task. A task cannot suspend itself. Suspend requests cannot be nested.

3.5.2 Resume Task (rsm_tsk) [T/D/L]

C-Language Interface:

ER ercd = rsm_tsk (ID tskid);

Assembler Interface:

```
JSR        @rsm_tsk
```

Parameters:

ID	tskid	R1	Task ID

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid ≤ 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (-H'34)	[p]	Undefined (Task indicated by tskid does not exist)
E_OBJ	H'ffc1 (–H'3f)	[k]	Object status is incorrect (Task indicated by tskid is not in SUSPEND state)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call rsm_tsk)

Description:

If the task specified by parameter tskid is in the SUSPEND state, the task that has been shifted to the SUSPEND state by system call sus_tsk is released from the SUSPEND state, and enters the READY state. In addition, a task in the WAIT-SUSPEND state is shifted to the WAIT state.

The suspend requests cannot be nested, so the task in the SUSPEND state is always released from the SUSPEND state by this system call.

A current task cannot cancel a suspend request for itself.

3.5.3 Sleep Task (slp_tsk) [T]
Sleep Task with Timeout (tslp_tsk) [T]

C-Language Interface:

ER ercd = slp_tsk (void);

ER ercd = tslp_tsk (TMO tmout);

Assembler Interface:

JSR	@slp_tsk
JSR	@tslp_tsk

Parameters:

TMO	tmout	ER2	Timeout specification <tslp_tsk>

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer driver cannot be used) (tslp_tsk)
E_PAR	H'ffdf (–H'21)	[p]	Invalid time specification (tmout ≤ –2) (tslp_tsk)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call slp_tsk)
		[k]	(A task portion issued system call slp_tsk or tslp_tsk while task dispatch was being disabled or while the CPU was being locked, or, in system call tslp_tsk, a type other than TMO_POL (0) was specified for parameter tmout.)
E_TMOUT	H'ffab (–H'55)	[k]	Timeout (tslp_tsk)

E_RLWAI	H'ffaa (–H'56)	[k]	WAIT state was forcibly cancelled (rel_wai system call was issued in WAIT state)

Description:

These system calls shift the current task from the RUN state to the WAIT state. A task can be released from the WAIT state by issuing system call wup_tsk and will terminate normally.

However, if system call wup_tsk, a wake-up request, has already been issued to the current task, the current task will not enter the WAIT state and will continue execution after decrementing the wake-up request count (wupcnt) by one.

Install the timer driver in the system to use the tslp_tsk system call. For details on how to install the timer driver, refer to section 6.2.1, Defining the Constant Definition Field.

The parameter tmout specified by system call tslp_tsk specifies the timeout period. If a positive number is specified for parameter tmout, error code E_TMOUT is returned when tmout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the task continues execution by decrementing the wake-up count by one if wupcnt has a positive number. If wake-up count is 0, error code E_TMOUT is returned.

If tmout = TMO_FEVR (–1) is specified, the same operation as for system call slp_tsk will be performed. In other words, timeout will not be monitored.

If system call sus_tsk is issued after a task has entered the WAIT state as a result of system call tslp_tsk, the task stays in the SUSPEND state even though the WAIT state has been released by system call wup_tsk, and the task will not resume execution until system call rsm_tsk is issued.

3.5.4 Wakeup Task (wup_tsk) [T/D/L]
Wakeup Task (iwup_tsk) [D/I]

C-Language Interface:

```
ER ercd = wup_tsk (ID tskid);

ER ercd = iwup_tsk (ID tskid);
```

Assembler Interface:

```
JSR    @wup_tsk

JSR    @iwup_tsk
```

Parameters:

ID	tskid	R1	Task ID

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid ≤ 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid does not exist)
E_OBJ	H'ffc1 (–H'3f)	[p]	Object status is incorrect (Current task is specified)
		[k]	(Task is in DORMANT state)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call iwup_tsk while tasks were being executed or a task-independent portion issued system call wup_tsk)
		[k]	(A task portion issued system call iwup_tsk while the CPU was being locked)
E_QOVR	H'ffb7 (–H'49)	[k]	Queue overflowed (wupcnt > H'ff)

Description:

These system calls release tasks from the WAIT state after the tasks were assigned to the WAIT state by system call slp_tsk or tslp_tsk. A task cannot wake up itself. If the task to be released from the WAIT state is not in the WAIT state (another task has not issued system call slp_tsk or tslp_tsk for that task), this wake-up request is queued and becomes valid the next time system call slp_tsk or tslp_tsk is issued for the specified task. Up to 255 (H'ff) wake-up requests (wupcnt) can be queued.

3.5.5 Cancel Wakeup Task (can_wup) [T/D/L]

C-Language Interface:

```
ER ercd = can_wup (INT *p_wupcnt, ID tskid);
```

Assembler Interface:

```
JSR        @can_wup
```

Parameters:

ID	tskid	R1	Task ID
INT	*p_wupcnt	---	Start address of the area where the number of queued wake-up requests are to be returned (C-language interface)

Return Parameters:

INT	*p_wupcnt	---	Start address of the area where the number of queued wake-up requests was stored (C-language interface)
---	wupcnt	R2	The number of queued task wake-up requests (Assembler interface)
ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (tskid < 0, tskid > Number of tasks defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Task indicated by tskid is not created)
E_OBJ	H'ffc1 (–H'3f)	[k]	Task is in DORMANT state
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call can_wup)

Description:

The system call can_wup releases all the queued task wake-up requests for the task specified by tskid, and returns the wake-up request count as the return parameter. If the queued task wake-up request is 0, 0 is returned.

A task that issued the system call can_wup can also be specified by setting tskid = TSK_SELF (0).

System call can_wup can be used to check if a task operation has been completed within the specified time or if the next task wake-up is being requested before system call slp_tsk for the previous task wake-up request has been issued. If the wake-up count request is not 0, the task has not been completed within the specified time. In that case, the user should take appropriate action to process the task.

3.6 Synchronization and Communication (Event Flag)

Event Flag System Calls: Event flags are controlled by the system calls listed in table 3.9.

Table 3.9 System Calls for Event Flag Control

System Call	Description	System State T/D/L/I
set_flg	Sets event flag	T/D/L
iset_flg	Sets event flag (dedicated to task-independent portion)	D/I
clr_flg	Clears event flag	T/D/L/I
wai_flg	Waits for event flag setting	T
pol_flg	Polls and waits for event flag	T/D/L/I
twai_flg	Waits for event flag (with timeout function)	T
ref_flg	Refers event flag state	T/D/L/I

Event Flag Specifications: The event flag specifications are listed in table 3.10.

Table 3.10 Event Flag Specifications

Item	Description
Event flag pattern size	16-bit size
Maximum number of event flags that can be defined	255
Event flag ID	1 to 255
Event flag initial value	0 (fixed value)
Event flag wait queue	The queue is managed on a first-in first-out (FIFO) basis and multiple tasks can wait for an event flag

Task-Execution Waiting and Release: Table 3.11 lists the causes of task-execution waiting and release.

Table 3.11 Causes of Task-Execution Waiting and Release

Cause of Waiting		Time of Release
When the current task enters the WAIT state	wai_flg or twai_flg system call	(1) When the event-flag wait condition is satisfied (2) When the specified timeout period (tmout) has passed (twai_flg) (3) When system call rel_wai is issued

3.6.1 Set Event Flag (set_flg) [T/D/L]
Set Event Flag (iset_flg) [D/I]

C-Language Interface:

ER ercd = set_flg (ID flgid, UINT setptn);

ER ercd = iset_flg (ID flgid, UINT setptn);

Assembler Interface:

JSR	@set_flg
JSR	@iset_flg

Parameters:

ID	flgid	R1	Event flag ID
UINT	setptn	R2	Bit pattern to set

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (flgid ≤ 0, flgid > Number of event flags defined)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call iset_flg while tasks were being executed or a task-independent portion issued system call set_flg)
		[k]	(A task portion issued system call iset_flg while the CPU was being locked)

Description:

These system calls perform a logical OR between the event-flag bits specified by flgid and the bit pattern specified by setptn, and set the result in the event-flag bits.

In these system calls, when a result of updating the event flag value satisfies the task wait cancellation conditions of the event flags, all tasks that satisfy the wait conditions are released from the wait state.

However, if a task is released from the WAIT state and TWF_CLR (clear) is specified for the task, the event flag bit pattern will be cleared, so later tasks will not be released from the WAIT state.

If all bits of setptn are zero, no operation is done to the event flag specified by flgid and the task terminates normally.

3.6.2 Clear Event Flag (clr_flg) [T/D/L/I]

C-Language Interface:

ER ercd = clr_flg (ID flgid, UINT setptn);

Assembler Interface:

JSR @clr_flg

Parameters:

ID	flgid	R1	Event flag ID
UINT	clrptn	R2	Bit pattern to clear

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (flgid ≤ 0, flgid > Number of event flags defined)

Description:

The system call clr_flg performs a logical AND between the event-flag bits specified by flgid (16 bits) and the bit pattern specified by clrptn and clears the event-flag bits whose corresponding bits specified by clrptn are zero.

Even if the event flag value has been changed in this system call, it does not release the tasks waiting for the event flag.

3.6.3 Wait for Eventflag (wai_flg) [T]
Wait for Eventflag (Polling) (pol_fig) [T/D/L/I]
Wait for Eventflag with Timeout(twai_fig) [T]

C-Language Interface:

ER ercd = wai_flg (UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);

ER ercd = pol_flg (UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);

ER ercd = twai_flg (UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode, TMO tmout);

Assembler Interface:

JSR	@wai_flg
JSR	@pol_flg
JSR	@twai_flg

Parameters:

UINT	*p_flgptn	---	Start address of the area where the bit pattern at wait release is to be returned (C-language interface)
ID	flgid	R1	Event flag ID
UINT	waiptn	R2	Wait bit pattern
UINT	wfmode	R3	Wait mode
TMO	tmout	ER4	Timeout specification <twai_flg>

Return Parameters:

UINT	*p_flgptn	---	Start address of the area where the bit pattern at wait release was stored (C-language interface)
---	flgptn	R2	Bit pattern at wait release (Assembler interface)
ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer driver and timeout function cannot be used) (twai_flg)
E_PAR	H'ffdf (–H'21)	[p]	Parameter error (waiptn = 0, wfmode is illegal)
		[p]	Invalid time specification (tmout ≤ –2) (twai_flg)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (flgid ≤ 0, flgid > Number of event flags defined)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call wai_flg or twai_flg)
		[k]	(A task portion issued system call wai_flg or twai_flg while task dispatch was being disabled or while the CPU was being locked, or, in system call twai_flg, a type other than TMO_POL (0) was specified for parameter tmout.)
E_RLWAI	H'ffaa (–H'56)	[k]	WAIT state was forcibly cancelled (rel_wai system call was issued in WAIT state)
E_TMOUT	H'ffab (–H'55)	[k]	Polling failed (pol_flg) Timeout (twai_flg)

Description:

A task that has issued one of these system calls waits until the event flags specified by the parameter flgid have been set according to the waiting conditions indicated by the parameters waiptn and wfmode.

The parameter wfmode can specify wait modes (table 3.12) in the following form.

wfmode:= (TWF_ANDW || TWF_ORW) [| TWF_CLR]

Table 3.12 Wait Modes (wfmode)

wfmode	Code	Description
TWF_ANDW	H'0000	AND wait
TWF_ORW	H'0002	OR wait
TWF_CLR	H'0001	Clear specification

If TWF_ANDW is specified as wfmode, the task waits until all the bits specified by waiptn have been set in the event flag specified by flgid. If TWF_ORW is specified as wfmode, the task waits until any one of the bits specified by waitpn has been set in the specified event flag. If TWF_CLR is specified, the event flag values (all bits) are cleared to 0 when the condition is satisfied and the task is released from the WAIT state. If the system call returns an error code, the value of the event flag will not be cleared. On the other hand, if TWF_CLR is not specified, the event flag value will not be cleared even if the condition is satisfied.

If the above conditions are satisfied before a task issues system call wai_flg or twai_flg, the task will terminate normally. If they are not satisfied, the task will be sent to the wait queue. Multiple tasks can wait for an event in the event flag queue.

The task issuing the system call pol_flg terminates normally if the event flag specified by flgid is set. If the event flag specified by flgid is not set, error code E_TMOUT will be returned.

System call twai_flg returns the value of the event flag to p_flgptn when the wait condition is satisfied. If TWF_CLR is specified, the value before the flag was cleared is returned.

The parameter tmout specified by system call twai_flg specifies the timeout period. If a positive number is specified for the parameter tmout, error code E_TMOUT is returned when the timeout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the task will not enter the WAIT state and will terminate normally if the event flag specified by flgid is set, or will return error code E_TMOUT if the event flag specified by flgid is not set. In other words, an operation the same as for the system call pol_flg will be performed.

If tmout = TMO_FEVR (–1) is specified, timeout watch is not performed. This means the same operation as for system call wai_flg will be performed.

If system call twai_flg is used, the timer driver must be installed in the system and (USE) must be specified for the timeout function in the setup table. For details on installing the timer driver and specifying the timeout function in the setup table, refer to section 6.2.1, Defining the Constant Definition Field.

3.6.4 Refer Event Flag State (ref_flg) [T/D/L/I]

C-Language Interface:

```
ER ercd = ref_flg (T_RFLG *pk_rflg, ID flgid);
```

Assembler Interface:

```
JSR @ref_flg
```

Parameters:

ID	flgid	R1	Event flag ID
T_RFLG	*pk_rflg	ER2/R2	Start address of the packet where the event flag state is to be returned

Return Parameters:

T_RFLG	*pk_rflg	ER2/R2	Start address of the packet where event flag state is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef    struct    t_rflg{
                     VP exinf;          0/0      4/2    Extended information
                     BOOL_ID wtsk;      +4/+2    2/2    Wait task ID
                     UINT flgptn;       +6/+4    2/2    Event flag bit pattern
                     }T_RFLG;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_rflg is 0 or an odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (flgid ≤ 0, flgid > Number of event flags defined)

Description:

The system call ref_flg refers to the state of the event flag (16 bits) indicated by the parameter flgid, and stores and returns extended information (exinf), a wait task ID (wtsk), and a current event-flag bit pattern (flgptn) to the area specified by pk_rflg. Note that an 8-byte (advanced mode) or 6-byte (normal mode) RAM area must be defined for the area specified by pk_rflg.

If there is no task waiting for the specified event flag, FALSE (0) is returned as a wait task ID.

If multiple tasks are waiting for the target event flag, the task ID at the head of the queue is returned as the wait task ID.

3.7 Synchronization and Communication (Semaphore)

Semaphore System Calls**:** Semaphores are controlled by the system calls listed in table 3.13.

Table 3.13 System Calls for Semaphore Control

System Call	Description	System State T/D/L/I
sig_sem	Returns semaphore resource	T/D/L
isig_sem	Returns semaphore resource (dedicated to task-independent portion)	D/I
wai_sem	Waits on semaphore	T
preq_sem	Polls and requests semaphore resource	T/D/L/I
twai_sem	Waits on semaphore with timeout	T
ref_sem	Refers semaphore state	T/D/L/I

Semaphore Specifications: The semaphore specifications are listed in table 3.14.

Table 3.14 Semaphore Specifications

Item	Description
Maximum semaphore count	65535
Maximum number of semaphores that can be defined	255
Semaphore ID	1 to 255
Semaphore counter initial value	1 (fixed value)
Semaphore wait task queue	The queue is managed on a first-in first-out (FIFO) basis and multiple tasks can wait for a semaphore

Task-Execution Waiting and Release: Table 3.15 lists the causes of task-execution waiting and release.

Table 3.15 Causes of Task-Execution Waiting and Release

Cause of Waiting		Time of Release
When the current task enter the WAIT state	wai_sem or twai_sem system call	(1) When the resource managed by semaphore is acquired (2) When the specified timeout period (tmout) has passed (twai_sem) (3) When system call rel_wai is issued

3.7.1 Returns Semaphore Resource (sig_sem) [T/D/L]
Returns Semaphore Resource (isig_sem) [D/I]

C-Language Interface:

ER ercd = sig_sem (ID semid);

ER ercd = isig_sem (ID semid);

Assembler Interface:

JSR	@sig_sem
JSR	@isig_sem

Parameters:

ID	semid	R1	Semaphore ID

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (semid ≤ 0, semid > Number of semaphores defined)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call isig_sem while tasks were being executed or a task-independent portion issued system call sig_sem)
		[k]	(A task portion issued system call isig_sem while the CPU was being locked)
E_QOVR	H'ffb7 (–H'49)	[k]	Queuing overflow (semcnt > H'ffff)

Description:

These system calls release the task at the head of a task wait queue from the WAIT state if there is a task waiting for the semaphore indicated by semid. If there are no tasks in a queue, the semaphore count (semcnt) is incremented by one.

3.7.2 Wait on Semaphore (wai_sem) [T]
Poll and Request Semaphore (preq_sem) [T/D/L/I]
Wait on Semaphore with Timeout (twai_sem) [T]

C-Language Interface:

```
ER ercd = wai_sem (ID semid);

ER ercd = preq_sem (ID semid);

ER ercd = twai_sem (ID semid, TMO tmout);
```

Assembler Interface:

```
JSR        @wai_sem

JSR        @preq_sem

JSR        @twai_sem
```

Parameters:

ID	semid	R1	Semaphore ID
TMO	tmout	ER4	Timeout specification <twai_sem>

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer driver and timeout function cannot be used) (twai_sem)
E_PAR	H'ffdf (–H'21)	[p]	Invalid time specification (tmout ≤ –2) (twai_sem)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (semid ≤ 0, semid > Number of semaphores defined)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call wai_sem or twai_sem)
		[k]	(A task portion issued system call wai_sem or twai_sem while task dispatch was being disabled or while the CPU was being locked, or, in system call twai_sem, a type other than TMO_POL (0) was specified for parameter tmout.)
E_RLWAI	H'ffaa (–H'56)	[k]	WAIT state was forcibly cancelled (rel_wai system call was issued in WAIT state)
E_TMOUT	H'ffab (–H'55)	[k]	Polling failed (preq_sem)
		[k]	Timeout (twai_sem)

Description:

These system calls decrement the count by one if the semaphore count specified by the parameter semid is equal to or greater than 1, and the task issuing the system calls continues execution.

If the semaphore count specified by semid is 1 or more for system call wai_sem or twai_sem, the count value is decremented by 1 and the task issuing the system call wai_sem or twai_sem terminates normally. If the semaphore count is 0 for system call wai_sem or twai_sem, the count value is not modified and the task issuing the system call wai_sem or twai_sem shifts to the WAIT state.

If the semaphore count specified by semid is 1 or more for system call preq_sem, the semaphore count value is decremented by 1 and the task issuing the system call preq_sem terminates normally. If the semaphore count is 0 for system call preq_sem, the semaphore count value is not modified and an error code is returned.

The parameter tmout specified by system call twai_sem specifies the timeout period. If a positive number is specified for the parameter tmout, error code E_TMOUT is returned when tmout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the task will not enter WAIT state and if the semaphore count specified by semid is 1 or more, the count value is decremented by 1 and the task terminates normally. If the semaphore count is 0, the count value is not modified and error code E_TMOUT is returned. In other words, the same operation as for the system call preq_sem will be performed.

If tmout = TMO_FEVR (–1) is specified, timeout watch is not performed. In other words, the same operation as for the system call wai_sem will be performed.

If system call twai_sem is used, the timer driver must be installed in the system and (USE) must be specified for the timeout function in the setup table. For details on installing the timer driver and specifying the timeout function in the setup table, refer to section 6.2.1, Defining the Constant Definition Field.

3.7.3 Refer Semaphore State (ref_sem) [T/D/L/I]

C-Language Interface:

ER ercd = ref_sem (T_RSEM *pk_rsem, ID semid);

Assembler Interface:

JSR	@ref_sem

Parameters:

ID	semid	R1	Semaphore ID
T_RSEM	*pk_rsem	ER2/R2	Start address of the packet where the semaphore state is to be returned

Return Parameters:

T_RSEM	*pk_rsem	ER2/R2	Start address of the packet where the semaphore state is stored
ER	ercd	R0	Error code

Packet Structure:

typedef struct t_rsem{

VP exinf;	0/0	4/2	Extended information
BOOL_ID wtsk;	+4/+2	2/2	If there is a task waiting (wait task ID)
UINT semcnt;	+6/+4	2/2	Current semaphore count value

}T_RSEM;

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_rsem is 0 or an odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (semid ≤ 0, semid > Number of semaphores defined)

Description:

The ref_sem system call refers to the state of the semaphore indicated by the parameter semid and stores and returns extended information (exinf), wait task ID (wtsk), and current semaphore count (semcnt) to the area specified by pk_rsem. Note that an 8-byte (advanced mode) or a 6-byte (normal mode) RAM area must be defined for the area specified by pk_rsem.

If there is no task waiting for the specified semaphore, FALSE (0) is returned as a wait task ID. If multiple tasks are waiting for the target semaphore, the task ID at the head of the queue is returned as the wait task ID.

3.8 Synchronization and Communication (Mailbox)

Mailbox System Calls: Mailboxes are controlled by the system calls listed in table 3.16.

Table 3.16 System Calls for Mailbox Control

System Call	Description	System State T/D/L/I
snd_msg	Sends message	T/D/L
isnd_msg	Sends message (dedicated to task-independent portion)	D/I
rcv_msg	Receives message from mailbox	T
prcv_msg	Polls and receives message	T/D/L/I
trcv_msg	Receives message from mailbox with timeout	T
ref_mbx	Refers mailbox state	T/D/L/I

Mailbox Specifications: The mailbox specifications are listed in table 3.17.

Table 3.17 Mailbox Specifications

Item	Description
Maximum number of mailboxes that can be defined	255
Mailbox ID	1 to 255
Message queue	The queue is managed on a first-in first-out basis (FIFO) and multiple tasks can wait for a message
Message	The first four bytes of a message are used by the kernel Before a message is sent, this area must be cleared to zero A message must be created in the RAM area

Task-Execution Waiting and Release: Table 3.18 lists the causes of task-execution waiting and release.

Table 3.18 Causes of Task-Execution Waiting and Release

Cause of Waiting		Time of Release
When the current task enters the WAIT state	rcv_msg or trcv_msg system call	(1) When a message is sent to the mailbox (2) When the specified timeout period (tmout) has passed (trcv_msg) (3) When system call rel_wai is issued

3.8.1 Send Message to Mailbox (snd_msg) [T/D/L]
Send Message to Mailbox (isnd_msg) [D/I]

C-Language Interface:

```
ER ercd = snd_msg (ID mbxid, T_MSG *pk_msg);

ER ercd = isnd_msg (ID mbxid, T_MSG *pk_msg);
```

Assembler Interface:

```
JSR    @snd_msg

JSR    @isnd_msg
```

Parameters:

ID	mbxid	R1	Mailbox ID
T_MSG	*pk_msg	ER2/R2	Start address of the message to send

Return Parameter:

ER	ercd	R0	Error code

Packet Structure:

Note: Since the T_MSG structure depends on the user, packets are not defined in the sample header file. Therefore, define them when necessary.

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (The message start address is 0 or an odd address)
		[k]	Invalid message form (The first four bytes of the message are not 0s)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mbxid ≤ 0, mbxid > Number of mailboxes defined)

E_CTX	H'ffbb (–H'45)	[p]	Context error (A task portion issued system call isnd_msg while tasks were being executed or a task-independent portion issued system call snd_msg)
		[k]	(A task portion issued system call isnd_msg while the CPU was being locked)

Description:

These system calls send a message specified by pk_msg to the mailbox specified by mbxid. The contents of the message are not copied to the mailbox; only the start address of the message (the value of pk_msg) is passed at message reception. Note, therefore, that if a message is modified after it has been sent by this system call, a task will not receive the correct message with the system call rcv_msg, prcv_msg, or trcv_msg.

If there is a task waiting to receive a message in the mailbox, the task at the head of the wait queue receives the message and is released from the WAIT state. On the other hand, if there are no tasks waiting to receive message, the message is placed in the mailbox and sent to the message queue.

The message area must be defined in a RAM area. Note that the user must control the message size because the kernel does not control it. The user can use only the area following the first four-byte area managed by the kernel. Since the kernel uses the first four bytes of the message, this four-byte area must initially be set to 0 and must not be changed even after the message transfer.

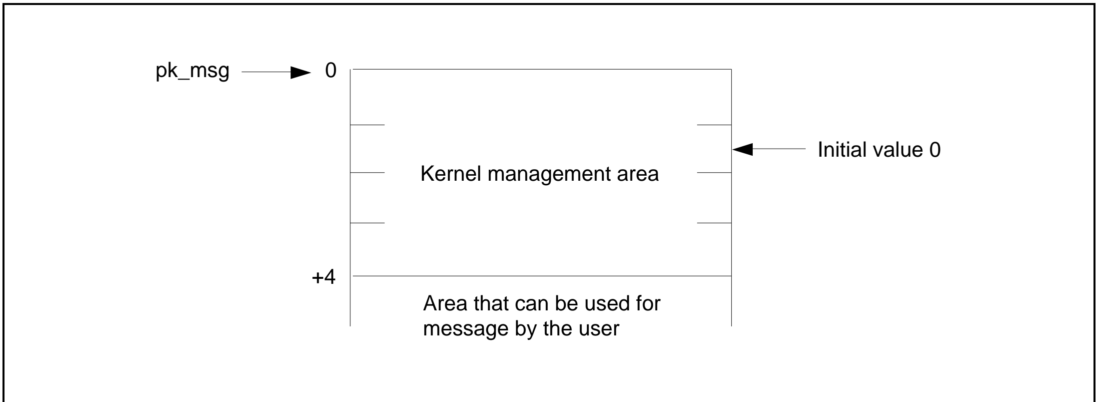


Figure 3.2 Message Form

3.8.2 Receive Message from Mailbox (rcv_msg) [T]
Poll and Receive Message from Mailbox (prcv_msg) [T/D/L/I]
Receive Message from Mailbox with Timeout (trcv_msg) [T]

C-Language Interface:

```
ER ercd = rcv_msg (T_MSG **ppk_msg, ID mbxid);

ER ercd = prcv_msg (T_MSG **ppk_msg, ID mbxid);

ER ercd = trcv_msg (T_MSG **ppk_msg, ID mbxid, TMO tmout);
```

Assembler Interface:

```
JSR     @rcv_msg

JSR     @prcv_msg

JSR     @trcv_msg
```

Parameters:

T_MSG	**ppk_msg	---	Start address of the area where the start address of the received message is to be returned (C-language interface)
ID	mbxid	R1	Mailbox ID
TMO	tmout	ER4	Timeout specification <trcv_msg>

Return Parameters:

T_MSG	**ppk_msg	---	Start address of the area where the received message was stored (C-language interface)
---	*pk_msg	ER2/R2	Start address of the received message (Assembler interface)
ER	ercd	R0	Error code

Packet Structure:

Note: Since the T_MSG structure depends on the user, packets are not defined in the sample header file. Therefore, define them when necessary.

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer driver and timeout function cannot be used) (trcv_msg)
E_PAR	H'ffdf (–H'21)	[p]	Invalid time specification (tmout ≤ –2) (trcv_msg)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mbxid ≤ 0, mbxid > Number of mailboxes defined)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call rcv_msg or trcv_msg)
		[k]	(A task portion issued system call rcv_msg or trcv_msg while task dispatch was being disabled or while the CPU was being locked, or, in system call trcv_msg, a type other than TMO_POL (0) was specified for parameter tmout.)
E_RLWAI	H'ffaa (–H'56)	[k]	WAIT state was forcibly cancelled (rel_wai system call was issued in WAIT state)
E_TMOUT	H'ffab (–H'55)	[k]	Polling failed (prcv_msg) Timeout (trcv_msg)

Description:

These system calls receive a message from the mailbox specified by the parameter mbxid. After the start address of the received message is stored in the area specified by the parameter ppk_msg, task execution continues.

With system calls rcv_msg and trcv_msg, if a message exists in the mailbox specified by mbxid, the start address of the message is stored in the area specified by ppk_msg and the task terminates normally. If there are no messages in the mailbox, the task that issued the system call rcv_msg or trcv_msg is placed in the task wait queue to receive a message. The task wait queue is managed on a first-in first-out (FIFO) basis.

With system call prcv_msg, if a message exists in the mailbox specified by mbxid, the start address of the message is stored in the area specified by ppk_msg and the task terminates normally. If there are no messages in the mailbox, error code E_TMOUT is returned.

The parameter tmout specified by system call trcv_msg specifies the timeout period. If a positive number is specified for the parameter tmout, error code E_TMOUT is returned when the timeout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the task will not enter the WAIT state, and if a message exists in the mailbox specified by mbxid, the start address of the message is stored in the area specified by ppk_msg and the task terminates normally. If there are no messages in the mailbox, error code E_TMOUT is returned. In other words, the same operation as for the system call prcv_msg will be performed.

If tmout = TMO_FEVR (–1) is specified, timeout watch is not performed. In other words, the same operation as for system call rcv_msg will be performed.

Note that a 4-byte RAM area is required for the area specified by ppk_msg.

Note: The user can use only the area following the first four-byte area managed by the kernel. Since the kernel uses the first four bytes of the message, this four-byte area must not be changed even after message transfer. If this area is rewritten before message is received after message has been sent, the system will not operate correctly.

If system call trcv_msg is used, the timer driver must be installed in the system and (USE) must be specified for the timeout function in the setup table. For details on installing the timer driver and specifying the timeout function in the setup table, refer to section 6.2.1, Defining the Constant Definition Field.

3.8.3 Refer Mailbox Status (ref_mbx) [T/D/L/I]

C-Language Interface:

ER ercd = ref_mbx (T_RMBX *pk_rmbx, ID mbxid);

Assembler Interface:

JSR @ref_mbx

Parameters:

ID	mbxid	R1	Mailbox ID
T_RMBX	*pk_rmbx	ER2/R2	Start address of the packet where the mailbox status is to be returned

Return Parameters:

T_RMBX	*pk_rmbx	ER2/R2	Start address of the packet where the mailbox status is stored
ER	ercd	R0	Error code

Packet Structure:

typedef struct t_rmbx{

VP exinf;	0/0	4/2	Extended information
BOOL_ID wtsk;	+4/+2	2/2	Wait task ID
T_MSG *pk_msg;	+6/+4	4/2	Start address of the message to receive next

}T_RMBX;

Note: Since the T_MSG structure depends on the user, it is not defined in the sample header file. Therefore, define them when necessary.

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_rmbx is 0 or an odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mbxid ≤ 0, mbxid > Number of mailboxes defined)

Description:

The ref_mbx system call refers to the status of the mailbox indicated by the parameter mbxid and stores and returns extended information (exinf), wait task ID (wtsk), and the start address of the message to be received next (pk_msg). A 10-byte (advanced mode) or a 6-byte (normal mode) RAM area is required for the area specified by pk_rmbx. If there are no tasks waiting to receive a message in the mailbox, FALSE (0) is returned as a wait task ID.

If multiple tasks are waiting for the target mailbox, the task ID at the head of the queue is returned as the wait task ID.

If there are no messages to receive next, NADR (–1) is returned as a message start address.

3.9 Interrupt Management

Interrupt Management System Calls: Interrupts are controlled by the system calls listed in table 3.19.

Table 3.19 System Calls for Interrupt Management

System Call	Description	System State T/D/L/I
ret_int	Returns from interrupt handler	I
chg_ims	Changes interrupt mask level	T/I
ref_ims	Refers interrupt mask level state	T/D/L/I
loc_cpu	Disables interrupt and dispatch	T/D/L
unl_cpu	Enables interrupt and dispatch	T/D/L

Interrupt Control Mode and Interrupt Mask Level: The kernel can be used in four interrupt control modes of the H8S series microcomputers.
The interrupt mask level of each interrupt control mode is shown in tables 3.20 to 3.23. For details on the interrupt control mode, CCR value, EXR value, or the interrupts that can be accepted, refer to the hardware manual of the CPU.

Table 3.20 Interrupt Mask Level in Interrupt Control Mode 0

Interrupt Mask Level (imask)	CCR		EXR			Acceptable Interrupts
	I	UI	I2	I1	I0	
1	1	—	—	—	—	NMI
0	0	—	—	—	—	All interrupts

Note: — indicates 0 or 1.

Table 3.21 Interrupt Mask Level in Interrupt Control Mode 1

Interrupt Mask Level (imask)	CCR		EXR			Acceptable Interrupts
	I	UI	I2	I1	I0	
3	1	1	—	—	—	NMI
2	1	0	—	—	—	Interrupts with control level 1
1	0	1	—	—	—	All interrupts
0	0	0	—	—	—	All interrupts

Note: — indicates 0 or 1.

Table 3.22 Interrupt Mask Level in Interrupt Control Mode 2

Interrupt Mask Level (imask)	CCR		EXR			Acceptable Interrupts
	I	UI	I2	I1	I0	
7	—	—	1	1	1	NMI
6	—	—	1	1	0	Interrupts with priority level 7
5	—	—	1	0	1	Interrupts with priority level 6 and 7
4	—	—	1	0	0	Interrupts with priority level 5 to 7
3	—	—	0	1	1	Interrupts with priority level 4 to 7
2	—	—	0	1	0	Interrupts with priority level 3 to 7
1	—	—	0	0	1	Interrupts with priority level 2 to 7
0	—	—	0	0	0	All interrupts

Note: — indicates 0 or 1.

Table 3.23 Interrupt Mask Level in Interrupt Control Mode 3

Interrupt Mask Level (imask)	CCR		EXR			Acceptable Interrupts
	I	UI	I2	I1	I0	
8	1	1	1	1	1	NMI
7	1	0	—	—	—	Interrupts with control level 1
6	0	0	1	1	0	Interrupts with priority level 7 and control level 0 or 1
5	0	0	1	0	1	Interrupts with priority level 6 and 7 and control level 0 or 1
4	0	0	1	0	0	Interrupts with priority level 5 to 7 and control level 0 or 1
3	0	0	0	1	1	Interrupts with priority level 4 to 7 and control level 0 or 1
2	0	0	0	1	0	Interrupts with priority level 3 to 7 and control level 0 or 1
1	0	0	0	0	1	Interrupts with priority level 2 to 7 and control level 0 or 1
0	0	0	0	0	0	All interrupts

Note: — indicates 0 or 1.

Note: In interrupt control mode 3 with the kernel interrupt mask level set to 7, the kernel interrupt with control level 1 cannot issue a system call because such an interrupt is considered to have an kernel interrupt mask level higher than 7; that is, it is masked.

3.9.1 Return from Interrupt Handler (ret_int) [I]

C-Language Interface:

None (ret_int can be issued by using the extended function #pragma interrupt of C compiler)

Assembler Interface:

```
JMP        @ret_int
```

Parameters:

None

Return Parameter:

None

Error Codes:

At normal termination:	[k]	Does not return to the task that issued this system call.
At abnormal termination:	[p]	If a task portion issues this system call while tasks are being executed, control is passed to the system termination routine.
	[k]	If a task portion issues this system call while the CPU is being locked, control is passed to the system termination routine.

Description:

The system call ret_int is used to return control from an interrupt handler. Even when a system call is issued for an interrupt handler, dispatch does not occur. Task dispatch is delayed until the system call ret_int has been issued to return control to the task from the interrupt handler.

Note: When issuing this system call, the contents of the stack pointer and registers must be the same as when the interrupt handler was initiated. Registers to be used by interrupt handler must be stored and restored by the user.

3.9.2 Change Interrupt Mask Level (chg_ims) [T/I]

C-Language Interface:

```
ER ercd = chg_ims (UINT imask);
```

Assembler Interface:

```
JSR        @chg_ims
```

Parameter:

UINT	imask	R1	Interrupt mask value Interrupt control mode 0: CR_IMS0 to CR_IMS1 (H'0 to H'1) Interrupt control mode 1: CR_IMS0 to CR_IMS3 (H'0 to H'3) Interrupt control mode 2: CR_IMS0 to CR_IMS7 (H'0 to H'7) Interrupt control mode 3: CR_IMS0 to CR_IMS8 (H'0 to H'8)

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Parameter error (imask outside the range)
E_CTX	H'ffbb (–H'45)	[k]	Context error (A task portion issued system call chg_ims while task dispatch was being disabled or while the CPU was being locked)

Description:

The system call chg_ims changes the current interrupt mask to the level specified by imask. Specify CR_IMSn (n: 0 to 8) for imask according to the interrupt control mode.

For an interrupt mask, interrupts can be inhibited or enabled by directly setting values CCR and EXR.

For details on interrupt mask value (imask) and the value of CCR or EXR, refer to tables 3.20, 3.21, 3.22, and 3.23.

If an interrupt is masked by this system call, the system makes a transition from the task portion execution to task-independent portion execution. Therefore, a system call that moves the task to the WAIT state or system calls dedicated to task portion cannot be issued.

To return execution from the task-independent portion to the task portion, the interrupt mask specified by this system call must also be cancelled by this system call. If a task switch is requested during the task-independent portion execution, task switch request is suspended until the interrupt mask of the task is changed to CR_IMS0(H'0) by this system call (execution returns to task portion to execute tasks).

Note: If the interrupt mask level is changed to the level exceeding the kernel interrupt mask level defined in the setup table, do not issue system calls other than the chg_ims system call which lowers the interrupt mask level equal to or less than the kernel interrupt mask level. If such system calls are issued, the system may not operate correctly.

3.9.3 Refer Interrupt Mask Level State (ref_ims) [T/D/L/I]

C-Language Interface:

```
ER ercd = ref_ims (UINT *p_imask);
```

Assembler Interface:

```
JSR @ref_ims
```

Parameters:

UINT	*p_imask	---	Start address of the area where the interrupt mask level is to be returned (C-language interface)

Return Parameters:

UINT	*p_imask	---	Start address of the area where the interrupt mask level was stored (C-language interface)
---	imask	R1	Interrupt mask level (Assembler interface)
ER	ercd	R0	Error code

Error Code:

E_OK	H'0000	[k]	Normal termination

Description:

The system call ref_ims returns the current interrupt mask level.
The value range and contents that can be used as the interrupt mask level depend on the interrupt control modes.

3.9.4 Lock CPU (loc_cpu) [T/D/L]

C-Language Interface:

```
ER ercd = loc_cpu (void);
```

Assembler Interface:

```
JSR        @loc_cpu
```

Parameters:

None

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call loc_cpu)

Description:

The system call loc_cpu locks the CPU and inhibits interrupts and task dispatches. To unlock the CPU and to execute other tasks, the system call unl_cpu must be issued.

The following indicates the characteristics of the while the CPU is being locked.

1. Since tasks cannot be scheduled while the CPU is being locked, tasks other than the current task cannot enter the RUN state. Tasks are scheduled again after the CPU has been unlocked.

2. While the CPU is being locked, interrupts, having a level equal to or below the kernel interrupt mask level defined in the setup table, are masked. Interrupts with levels equal to or lower than this level cannot be accepted.

3. System calls that shifts a task to WAIT state cannot be issued.

Issuing the system call dis_dsp disables task dispatch, and issuing system call loc_cpu while task dispatch is disabled locks the CPU. The system can make a transition from the dispatch-disabled state to the CPU-locked state. However, system cannot make a transition from the CPU-locked state to the dispatch-disabled state. If system call ena_dsp is issued to enable task dispatch while the CPU is being locked, error code E_CTX is returned. If the system call ext_tsk is issued to terminate the task that has been monopolizing the CPU, other tasks will be executed again.

If the system call loc_cpu is issued while the CPU is being locked, an error will not occur. In this case, queuing will not be performed.

Table 3.24 shows system transition. In this table, the numbers in each system call column show the state number to shift to. For example, if the system call dis_dsp is issued at state number 1 (task-execution state), the task enters state number 2. E_CTX is an error code, which is returned as a result of issuing the corresponding system call.

Table 3.24 State Transition by Issuing dis_dsp, ena_dsp, loc_cpu, and unl_cpu

State Number	System State	Current State		System Call to Issue			
		Interrupt	Dispatch	dis_dsp	ena_dsp	loc_cpu	unl_cpu
1	Task-execution state (TSS_TSK)	Enabled	Enabled	Shifts to 2	Shifts to 1	Shifts to 3	Shifts to 1
2	Dispatch-disabled state (TSS_DDSP)	Enabled	Disabled	Shifts to 2	Shifts to 1	Shifts to 3	Shifts to 1
3	CPU locked-state (TSS_LOC)	Disabled	Disabled	If issued, E_CTX is returned	If issued, E_CTX is returned	Shifts to 3	Shifts to 1

Notes: 1. The interrupt mask level used while the CPU is being locked is the kernel interrupt mask level defined in the setup table. Therefore, interrupts with a level higher than the kernel interrupt mask level can be accepted while the CPU is being locked.

2. When task dispatch is disabled or the CPU is locked, do not change the interrupt mask level by directly changing the register value; otherwise normal system operation cannot be guaranteed.

3.9.5 Unlock CPU (unl_cpu) [T/D/L]

C-Language Interface:

ER ercd = unl_cpu (void);

Assembler Interface:

```
JSR        @unl_cpu
```

Parameters:

None

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call unl_cpu)

Description:

The system call unl_cpu permits interrupts and task dispatches; it unlocks the CPU, which was locked by system call loc_cpu, to execute other tasks. Then task dispatch (scheduling) is performed.

The system call unl_cpu is usually used to unlock the CPU to execute other tasks; however, the same process can be performed by issuing the system call while task dispatch is being disabled.

If the system call unl_cpu is issued while tasks are being executed, an error will not occur. In that case, queuing will not be performed.

3.10 Memory Pool Management (Fixed-Size Memory Pool)

Fixed-Size Memory Pool System Calls: Fixed-size memory pools are controlled by the system calls listed in table 3.25.

Table 3.25 System Calls for Fixed-Size Memory Pool Control

		System State
System Call	**Description**	**T/D/L/I**
get_blf	Gets fixed-size memory block	T
pget_blf	Polls and gets fixed-size memory block	T/D/L/I
tget_blf	Gets fixed-size memory block with timeout function	T
rel_blf	Releases fixed-size memory block	T/D/L
ref_mpf	Refers fixed-size memory pool state	T/D/L/I

Fixed-Size Memory Pool Specifications: The fixed-size memory pool specifications are listed in table 3.26.

Table 3.26 Fixed-Size Memory Pool Specifications

Item	Description
Maximum number of fixed-size memory pools that can be defined	255
Fixed-size memory pool ID	1 to 255 (including undefined memory pools)
Number of memory blocks	65535
Memory block size	2 to 65530
Memory block wait queue	The queue is managed on a first-in first-out (FIFO) basis and multiple tasks can wait for a fixed-size memory block

Task-Execution Waiting and Release: Table 3.27 lists the causes of task-execution waiting and release.

Table 3.27 Causes of Task-Execution Waiting and Release

Cause of Waiting		Time of Release
When the current task enters the WAIT state	get_blf or tget_blf system call	(1) When a memory block is acquired (2) When the specified timeout period (tmout) has passed (tget_blf) (3) When system call rel_wai is issued

3.10.1 Get Fixed-Size Memory Block (get_blf) [T]
Poll and Get Fixed-Size Memory Block (pget_blf) [T/D/L/I]
Get Fixed-Size Memory Block with Timeout (tget_blf) [T]

C-Language Interface:

```
ER ercd = get_blf (VP *p_blf, ID mpfid);

ER ercd = pget_blf (VP *p_blf, ID mpfid);

ER ercd = tget_blf (VP *p_blf, ID mpfid, TMO tmout);
```

Assembler Interface:

```
JSR     @get_blf

JSR     @pget_blf

JSR     @tget_blf
```

Parameters:

VP	*p_blf	---	Start address of the area where the start address of the memory block is to be returned (C-language interface)
ID	mpfid	R1	Fixed-size memory pool ID
TMO	tmout	ER4	Timeout specification <tget_blf>

Return Parameters:

VP	*p_blf	---	Start address of the area where the start address of the memory block was stored (C-language interface)
---	blf	ER2/R2	Memory block start address (Assembler interface)
ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer driver and timeout function cannot be used) (tget_blf)
E_PAR	H'ffdf (–H'21)	[p]	Parameter error (tmout ≤ –2) (tget_blf)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mpfid ≤ 0, mpfid > Number of memory pools defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Fixed-size memory pool indicated by mpfid does not exist)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call get_blf or tget_blf)
		[k]	(A task portion issued system call get_blf or tget_blf while task dispatch was being disabled or while the CPU was being locked, or, in system call tget_blf, a type other than TMO_POL (0) was specified for parameter tmout.)
E_RLWAI	H'ffaa (–H'56)	[k]	WAIT state was forcibly cancelled (rel_wai system call was issued in WAIT state)
E_TMOUT	H'ffab (–H'55)	[k]	Polling failed (pget_blf) Timeout (tget_blf)

Description:

These system calls get one fixed-size memory block from the fixed-size memory pool indicated by mpfid. After the start address of the acquired memory block is stored in the area specified by p_blf, task execution continues.

With system call get_blf or tget_blf, if a memory block is available in the fixed-size memory pool specified by mpfid, the start address of a memory block is stored in the area specified by p_blf and the task terminates normally. If there is a task already waiting for a memory block, or if no task is waiting but there is no memory block available in the fixed-size memory pool, the task having issued the system call get_blf or tget_blf is placed in the task wait queue until a memory block can be acquired. The queue is managed on a first-in first-out (FIFO) basis.

With system call pget_blf, if a memory block is available in the memory pool specified by mpfid, the start address of a memory block is stored in the area specified by p_blf and the task terminates normally. If a task is already waiting to get a memory block, or if no task is waiting but there is no memory block available in the fixed-size memory pool, error code E_TMOUT is returned.

The parameter tmout of system call tget_blf specifies the timeout period. If a positive number is specified for the parameter tmout, the error code E_TMOUT is returned when the timeout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the task will not enter the WAIT state, and if a memory block is available in the fixed-size memory pool specified by mpfid, the start address of the memory block is stored in the area specified by p_blf and the task terminates normally. If a task is already waiting to get a memory block, or if no task is waiting but there is no memory block available in the fixed-size memory pool, error code E_TMOUT is returned. In other words, the same operation as for the system call pget_blf will be performed.

If tmout = TMO_FEVR (–1) is specified, timeout watch is not performed. In other words, the same operation as for the system call get_blf will be performed.

After the memory block has been acquired, the size of the fixed-size memory pool free space will decrease by the size calculated in the following expression:

Decrease in size = Block size specified at memory pool creation + 4 bytes

If system call tget_blf is used, the timer driver must be installed in the system and (USE) must be specified for the timeout function in the setup table. For details on installing the timer driver and specifying the timeout function in the setup table, refer to section 6.2.1, Defining the Constant Definition Field.

3.10.2 Release Fixed-Size Memory Block (rel_blf) [T/D/L]

C-Language Interface:

ER ercd = rel_blf (ID mpfid, VP blf);

Assembler Interface:

JSR @rel_blf

Parameters:

ID	mpfid	R1	Fixed-size memory pool ID
VP	blf	ER2/R2	Start address of memory block

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (blf is 0 or an odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mpfid ≤ 0, mpfid > Number of memory pools defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Fixed-size memory pool indicated by mpfid does not exist)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call rel_blf)
EV_ILBLK	H'ffle (–H'e2)	[k]	Invalid memory block (blf is other than the memory pool area or blf has already been returned)

Description:

The system call rel_blf returns the memory block start address indicated by blf to the fixed-size memory pool indicated by mpfid.

The start address of the memory block acquired by the system call get_blf, pget_blf, or tget_blf is specified by parameter blf.

If there is a task waiting to get a memory block, the return address is passed to the task at the head of the task wait queue, releasing it from WAIT state.

3.10.3 Refer Fixed-Size Memory Pool Status (ref_mpf) [T/D/L/I]

C-Language Interface:

```
ER ercd = ref_mpf (T_RMPF *pk_rmpf, ID mpfid);
```

Assembler Interface:

```
JSR        @ref_mpf
```

Parameters:

ID	mpfid	R1	Fixed-size memory pool ID
T_RMPF	*pk_rmpf	ER2/R2	Start address of the packet where the fixed-size memory pool status is to be returned

Return Parameters:

T_RMPF	*pk_rmpf	ER2/R2	Start address of the packet where the fixed-size memory pool status is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef    struct    t_rmpf{
                     VP exinf;        0/0      4/2   Extended information
                     BOOL_ID wtsk;    +4/+2    2/2   Wait task ID
                     INT frbcnt;      +6/+4    2/2   Number of blocks of memory space
                                                     available
                     INT mpfcnt;      +8/+6    2/2   Number of blocks of the memory pool
                     INT blfsz;       +10/+8   2/2   Fixed-size memory block size (Number
                                                     of bytes)
                     }T_RMPF;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_rmpf is 0 or an odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mpfid ≤ 0, mpfid > Number of memory pools defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Fixed-size memory pool indicated by mpfid does not exist)

Description:

The system call ref_mpf refers to the status of the fixed-size memory pool indicated by mpfid and stores and returns extended information (exinf), wait task ID (wtsk), number of blocks of memory space available (frbcnt), number of blocks of memory pool (mpfcnt), and fixed-size memory block size (blfsz) to the area specified by pk_rmpf. A 12-byte (advanced mode) or 10-byte (normal mode) RAM area is required for the area specified by pk_rmpf. If there is no task waiting for the specified memory pool cannot provide the memory pool immediately, FALSE (0) is returned as a wait task ID.

If multiple tasks are waiting for the target memory pool, the task ID of the task at the head of the wait queue is returned as the wait task ID.

3.11 Memory Pool Management (Variable-Size Memory Pool)

Variable-Size Memory Pool System Calls: Variable-size memory pools are controlled by the system calls listed in table 3.28.

Table 3.28 System Calls for Variable-Size Memory Pool Control

System Call	Description	System State T/D/L/I
get_blk	Gets variable-size memory block	T
pget_blk	Polls and gets variable-size memory block	T/D/L/I
tget_blk	Gets variable-size memory block with timeout	T
rel_blk	Returns variable-size memory block	T/D/L
ref_mpl	Refers variable-size memory pool status	T/D/L/I

Variable-size Pool Specifications: The variable-size memory pool specifications are listed in table 3.29.

Table 3.29 Variable-Size Memory Pool Specifications

Item	Description
Maximum number of variable-size memory pools that can be defined	255
Variable-size memory pool ID	1 to 255
Memory block wait queue	The queue is managed on a first-in first-out (FIFO) basis and multiple tasks can wait for variable-size memory blocks

Task-Execution Waiting and Release: Table 3.30 lists the causes of task-execution waiting and release.

Table 3.30 Causes of Task-Execution Waiting and Release

Cause of Waiting		Time of Release
When the current task enters the WAIT state	get_blk or tget_blk system call	(1) When a memory block is acquired (2) When the specified timeout period (tmout) has passed (tget_blk) (3) When system call rel_wai is issued

Fragmentation of Variable-Size Memory Pool: Repeated acquisition and return of memory blocks from the variable-size memory pool causes fragmentation of the available memory area in the memory pool, thus resulting in a smaller maximum available contiguous memory area. When there is a memory block that is not to be returned, the size of the maximum available contiguous memory area will never be larger than a certain size because such a block behaves as a barrier. However, the kernel cannot de-fragment memory area. To avoid this problem, get a memory block that is not to be returned right after a memory pool is created, that is, before any memory block to be returned is acquired.

3.11.1 Get Variable-Size Memory Block (get_blk) [T]
Poll and Get Variable-Size Memory Block (pget_blk) [T/D/L/I]
Get Variable-Size Memory Block with Timeout (tget_blk) [T]

C-Language Interface:

```
ER ercd = get_blk (VP *p_blk, ID mplid, UW blksz);

ER ercd = pget_blk (VP *p_blk, ID mplid, UW blksz);

ER ercd = tget_blk (VP *p_blk, ID mplid, UW blksz, TMO tmout);
```

Assembler Interface:

```
JSR    @get_blk

JSR    @pget_blk

JSR    @tget_blk
```

Parameters:

VP	*p_blk	---	Start address of the area where the start address of the memory block is to be returned (C-language interface)
ID	mplid	R1	Variable-size memory pool ID
UW	blksz	ER2	Memory block size (Number of bytes)
TMO	tmout	ER4	Timeout specification <tget_blk>

Return Parameters:

VP	*p_blk	---	Start address of the area where the start address of the memory block was stored (C-language interface)
---	blk	ER2/R2	Memory block start address (Assembler interface)
ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer driver and timeout function cannot be used) (tget_blk)
E_PAR	H'ffdf (–H'21)	[p]	Parameter error (blksz is 0 or and odd address, mplsz < blksz) (tmout ≤ –2) (tget_blk)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mplid ≤ 0, mplid > Number of memory pools defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Variable-size memory pool indicated by mplid does not exist)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call get_blk or tget_blk)
		[k]	(A task portion issued system call get_blk or tget_blk while task dispatch was being disabled or while the CPU was being locked, or, in system call tget_blk , a type other than TMO_POL (0) was specified for parameter tmout.)
E_RLWAI	H'ffaa (–H'56)	[k]	WAIT state was forcibly cancelled (rel_wai system call was issued in WAIT state)
E_TMOUT	H'ffab (–H'55)	[k]	Polling failed (pget_blk) Timeout (tget_blk)

Description:

These system calls get memory blocks if the variable-size memory pool specified by mplid has the memory size specified by blksz. After the start address of the acquired memory block is stored in the area specified by p_blk, task execution continues. Note that the size of the variable-size memory pool specified by mplid must be equal to or more than (blksz + 16) bytes for the task to get the memory block because additional 16 bytes are required for OS management purposes.

With system calls get_blk and tget_blk, if the variable-size memory pool specified by mplid has the memory size specified by blksz, the start address of the memory block is stored in the area specified by p_blk and the task terminates normally. Note that the size of the variable-size memory pool specified by mplid must be equal to or more than (blksz + 16) bytes for the task to get the memory block because additional 16 bytes are required for OS management purposes. If there is a task already waiting for a memory block, or if no task is waiting but the available memory size is less than the size specified by blksz (which means that the available memory size is less than (blksz + 16) bytes), the task having issued the system call get_blk or tget_blk is placed into the task queue until memory can be acquired. The queue is managed on a first-in first-out (FIFO) basis.

With system call pget_blk, if the variable-size memory pool specified by mplid has the memory size specified by blksz, the start address of the memory block is stored in the area specified by p_blk and the task terminates normally. Note that the size of the variable-size memory pool specified by mplid must be equal to or more than (blksz + 16) bytes for the task to get the memory block because additional 16 bytes are required for OS management purposes. If there is a task already waiting for a memory block, or if no task is waiting but the available memory size is less than the size specified by blksz (which means that the available memory size is less than (blksz + 16) bytes), error code E_TMOUT is returned.

The parameter tmout specified by system call tget_blk specifies this wait period. If a positive number is specified for parameter tmout, error code E_TMOUT is returned when the timeout period has passed without the wait release conditions being satisfied.

If tmout = TMO_POL (0) is specified, the task will not enter the WAIT state, and if the variable-size memory pool specified by mplid has the memory size specified by blksz, the start address of the memory block is stored in the area specified by p_blk and the task terminates normally. Note that the size of the variable-size memory pool specified by mplid must be equal to or more than (blksz + 16) bytes for the task to get the memory block because additional 16 bytes are required for OS management purposes. If there is a task already waiting for a memory block, or if no task is waiting but the available memory size is less than the size specified by blksz (which means that the available memory size is less than (blksz + 16) bytes), error code E_TMOUT is returned. In other words, the same operation as for the system call pget_blk will be performed.

If tmout = TMO_FEVR (–1) is specified, timeout watch is not performed. In other words, the same operation as for the system call get_blk will be performed.

After the memory block has been acquired, the size of the variable-size memory pool free space will decrease by the size calculated in the following expression:

Decrease in size = blksz + 16 bytes

If system call tget_blk is used, the timer driver must be installed in the system and (USE) must be specified for the timeout function in the setup table. For details on installing the timer driver and specifying the timeout function in the setup table, refer to section 6.2.1, Defining the Constant Definition Field.

3.11.2 Release Variable-Size Memory Block (rel_blk) [T/D/L]

C-Language Interface:

```
ER ercd = rel_blk (ID mplid, VP blk);
```

Assembler Interface:

```
JSR        @rel_blk
```

Parameters:

ID	mplid	R1	Variable-size memory pool ID
VP	blk	ER2/R2	Start address of memory block

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (blk is 0 or odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mplid ≤ 0, mplid > Number of memory pools defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Variable-size memory pool indicated by mplid does not exist)
E_CTX	H'ffbb (–H'45)	[p]	Context error (A task-independent portion issued system call rel_blk)
EV_ILBLK	H'ffle (–H'e2)	[k]	Invalid memory block (blk is other than the memory pool area or blk has already been returned)

Description:

The system call rel_blk returns the memory block start address specified by blk to the variable-size memory pool specified by mplid.

If there is a task waiting to get a memory block from the variable-size memory pool indicated by mplid, and if the task at the head of the queue can get a memory block due to a memory block being returned to the memory pool, then that memory block start address is assigned to the task, releasing the task from the WAIT state.

If multiple tasks are waiting to get a memory block from the variable-size memory pool indicated by mplid, and if multiple tasks of the queue can get a memory block due to a memory block being returned to the memory pool, then block start addresses are assigned to tasks in the queue starting from the task at the head of the queue, releasing them from the WAIT state until the requested memory size can no longer be acquired.

The parameter blk specifies the start address of the memory block acquired by the system call get_blk, pget_blk, or tget_blk.

If blk is not the start address, or is the start address of the memory block that has already been returned, EV_ILBLK is returned as an error code.

3.11.3 Refer Variable-Size Memory Pool Status (ref_mpl) [T/D/L/I]

C-Language Interface:

```
ER ercd = ref_mpl (T_RMPL *pk_rmpl, ID mplid);
```

Assembler Interface:

```
JSR        @ref_mpl
```

Parameters:

ID	mplid	R1	Variable-size memory pool ID
T_RMPL	*pk_rmpl	ER2/R2	Start address of the packet where the variable-size memory pool status is to be returned

Return Parameters:

T_RMPL	*pk_rmpl	ER2/R2	Start address of the packet where the variable-size memory pool status is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef    struct    t_rmpl{
                     VP exinf;          0/0       4/2    Extended information
                     BOOL_ID wtsk;      +4/+2     2/2    Wait task ID
                     UW frsz;           +6/+4     4/4    Total size of available memory area
                     UW maxsz;          +10/+8    4/4    Maximum memory area available
                     UW mplsz;          +14/+12   4/4    Memory pool size
                     }T_RMPL;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_rmpl is 0 or on odd address)
E_ID	H'ffdd (–H'23)	[p]	Invalid ID number (mplid ≤ 0, mplid > Number of memory pools defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Variable-size memory pool indicated by mplid does not exist)

Description:

The system call ref_mpl refers to the status of the memory pool indicated by mplid and stores and returns extended information (exinf), wait task ID (wtsk), current free memory area total size (frsz), maximum free memory space size (maxsz), and memory pool size (mplsz) to the area specified by pk_rmpl.

An 18-byte (advanced mode) or 16-byte (normal mode) RAM area is required for the area specified by pk_rmpl.

The current total size of memory means the total size of free memory areas scattered in the memory pool (fragmented). The maximum free memory space means the maximum size of consecutive free memory areas scattered in the memory pool (fragmented).

Note that the maximum free memory space contains kernel management area (16 bytes), which is required each time a system call get_blk, pget_blk, or tget_blk is issued. In other words, it means the largest blksz that can be acquired immediately by issuing system call get_blk, pget_blk, or tget_blk.

If there is no task waiting to get a memory block, FALSE (0) is returned as a wait task ID.

If multiple tasks are waiting to get a memory block-size from the variable-size memory pool, the task ID of the task at the head of the queue is returned as the wait task ID.

If the specified memory pool cannot provide the memory block immediately, FALSE (0) is returned as a maximum memory space.

3.12 Time Management

Time Management System Calls: The time management function controls the system clock and cyclic handlers. Time is managed by the system calls listed in table 3.31 and cyclic handlers are controlled by the system calls listed in table 3.32.

Table 3.31 System Calls Related to the System Clock Control

System Call	Description	System State T/D/L/I
set_tim	Sets system clock	T/D/L/I
get_tim	Gets system clock	T/D/L/I

Table 3.32 System Calls for Cyclic Handler Control

System Call	Description	System State T/D/L/I
act_cyc	Controls cyclic handler activity	T/D/L/I
ref_cyc	Refers cyclic handler state	T/D/L/I

Time Management Specifications: The system clock specifications is listed in table 3.33 and the cyclic handler specifications are listed in table 3.34.

Table 3.33 System Clock Specifications

Item	Description
Clock value	Signed 48 bits
Clock initial value (the value at initialization)	H'000000000000

Table 3.34 Cyclic Handler Specifications

Item	Description
Maximum number of cyclic handlers that can be defined	255
Cyclic handler specification number	1 to 255
Cyclic handler initial activation state	TCY_ON (1) or TCY_OFF (0) (whichever specified in the setup table)
Cyclic time interval	H'1 to H'7FFFFFFF

Operating the Time Management: When using the time management function, the timer driver must be created and installed into the system every time a hardware timer interrupt occurs.

The following are performed in the kernel timer interrupt processing.

1. System clock is modified (+1).
2. All cyclic handlers that reached the cycle time are initiated and executed.
3. Timeout processing is performed by issuing system calls with timeout function.

These processes from 1 to 3 are performed with the timer interrupt level masked. Among these processes, 2 and 3 may overlap for multiple tasks and handlers. In that case, the processing time becomes very long and results in the following defects.

- Delay of the response to interrupts
- Delay of system clocks

To avoid these problems, the following steps must be taken:

- Do not shorten the timer interrupt cycle excessively.
- The timer handler processing time must be as short as possible.
- The timer handler cycle and the timeout value specified by the timeout system call must be set to a value as large as possible. For example, when the cycle time of a cyclic handler is 1 and the handler's processing time takes more time than the timer cycle time, that cyclic handler will be repeated infinitely, and the system will be hung.

Time Watch Method: The kernel manages the time watch for timeout system calls, cyclic handlers, and time management by the system clock using the relative time from the time of request. Therefore, the previous time watch request is not affected even if the system clock has been modified by the system call set_tim.

3.12.1 Set System Clock (set_tim) [T/D/L/I]

C-Language Interface:

```
ER ercd = set_tim (SYSTIME *pk_tim);
```

Assembler Interface:

```
JSR        @set_tim
```

Parameters:

SYSTIME	*pk_tim	ER2/R2	Start address of the packet where the current time data is indicated

Return Parameter:

ER	ercd	R0	Error code

Packet Structure:

```
typedef  struct  systime {
                 H utime;     0    2    Current time data (upper)
                 UH mtime;    +2   2    Current time data (middle)
                 UH ltime;    +4   2    Current time data (lower)
                 }SYSTIME;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer function cannot be used)
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_tim is 0 or an odd address)
			Invalid time specification (Value specified by pk_tim is negative)

Description:

The system call set_tim changes the current system clock retained in the system to a value specified by pk_tim. The number of bits allocated to the system clock is 48 (i.e., 16-bit utime, 16-bit mtime, and 16-bit ltime).

Note that the timeout period of a task and a timer handler (cyclic handler) that is being monitored cannot be modified with this system call. Therefore, a timeout error cannot occur until a certain amount of time has passed since the system call set_tim was issued.

3.12.2 Get System Clock (get_tim) [T/D/L/I]

C-Language Interface:

```
ER ercd = get_tim (SYSTIME *pk_tim);
```

Assembler Interface:

```
JSR        @get_tim
```

Parameters:

SYSTIME	*pk_tim	ER2/R2	Start address of the packet where the current time data is to be returned

Return Parameters:

SYSTIME	*pk_tim	ER2/R2	Start address of the packet where the current time data is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef   struct   systime {
                   H utime;    0    2    Current time data (upper)
                   UH mtime;   +2   2    Current time data (middle)
                   UH ltime;   +4   2    Current time data (lower)
                   }SYSTIME;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (–H'14)	[p]	Unsupported function (Timer function cannot be used)
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_tim is 0 or an odd address)

Description:

The system call get_tim reads the current system clock and returns it to the 6-byte RAM area indicated by pk_tim. The number of bits allocated to the system clock is 48 (i.e., 16-bit utime, 16-bit mtime, and 16-bit ltime).

3.12.3 Activate Cyclic Handler (act_cyc) [T/D/L/I]

C-Language Interface:

ER ercd = act_cyc (HNO cycno, UINT cycact);

Parameters:

HNO	cycno	R1	Cyclic handler number
UINT	cycact	R2	Activation state of the cyclic handler

Return Parameter:

ER	ercd	R0	Error code

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec (-H'14)	[p]	Unsupported function (Timer function cannot be used)
E_PAR	H'ffdf (-H'21)	[p]	Parameter error (cycact is illegal) cycno out of range: cycno ≤ 0, cycno > Number of cyclic handlers defined
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Cyclic handler specified by cycno is undefined)

Description:

The system call act_cyc changes the activation state of the cyclic handler indicated by the parameter cycno to the state indicated by the parameter cycact. The parameter cycact specifies the handler activation state (table 3.35) in the following format.

cycact:= (TCY_OFF || TCY_ON) [| TCY_INI]

Table 3.35 Handler Activation State (cycact)

cycact	Code	Description
TCY_OFF	H'0000	The cyclic handler is not initiated
TCY_ON	H'0001	The cyclic handler is initiated
TCY_INI	H'0002	The cyclic handler count is initialized (reset)

If TCY_OFF is specified for cycact, the cyclic handler activation state is turned off. Therefore, the cyclic handler will not be initiated even after a specified (cycle) time has passed. However, even when the activation state is off, a cycle time count is performed.

If TCY_ON is specified for cycact, the cyclic handler activation state is turned on. Since a cycle time count is performed even when the activation state is off, caution is needed because the length of time after the activation state has been turned on until the cyclic handler is initiated is undefined. However, when cycact = (TCY_ON | TCY_INI) is specified, the cyclic handler is initiated after the specified time has passed.

3.12.4 Refer Cyclic Handler State (ref_cyc) [T/D/L/I]

C-Language Interface:

```
ER ercd = ref_cyc (T_RCYC *pk_rcyc, HNO cycno);
```

Assembler Interface:

```
JSR        @ref_cyc
```

Parameters:

HNO	cycno	R1	Cyclic handler specification number
T_RCYC	*pk_rcyc	ER2/R2	Start address of the packet where the cyclic handler state is to be returned

Return Parameters:

T_RCYC	*pk_rcyc	ER2/R2	Start address of the packet where the cyclic handler state is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef   struct   t_rcyc{
                   VP exinf;           0/0       4/2   Extended information
                   CYCTIME lfttim;     +4/+2     4/4   Remaining time until the cyclic
                                                       handler is initiated
                   UINT cycact;        +8/+6     2/2   Cyclic handler activation state
                   FP cychdr;          +10/+8    4/2   Cyclic handler address
                   CYCTIME cyctim;     +14/+10   4/4   Cyclic timer interval
                   }T_RCYC;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_RSFN	H'ffec(-H'14)	[p]	Unsupported function (Timer driver cannot be used)
E_PAR	H'ffdf (-H'21)	[p]	Invalid address (pk_rcyc is 0 or an odd address)
			Parameter error (cycno ≤ 0, cycno > Number of cyclic handlers defined)
E_NOEXS	H'ffcc (–H'34)	[p]	Undefined (Cyclic handler specified by cycno is undefined)

Description:

The system call ref_cyc reads the cyclic handler state indicated by cycno and returns the extended information (exinf), remaining time until the cyclic handler is initiated (lfttim), cyclic handler activation state (cycact), cyclic handler address (cychdr), and cyclic timer interval (cyctim) in the area specified by the parameter pk_rcyc.

Note that an 18-byte (advanced mode) or a 14-byte (normal mode) RAM area must be defined for the area specified by the parameter pk_rcyc.

For the cyclic handler activation state (specified by cycact), only the information TCY_ON (H'0001) and TCY_OFF (H'0000) is returned; the information of TCY_INI (H'0002) is not returned.

3.13 System Management

3.13.1 get_ver (Get Version Information) [T/D/L/I]

C-Language Interface:

```
ER ercd = get_ver (T_VER *pk_ver);
```

Assembler Interface:

```
JSR          @get_ver
```

Parameters:

T_VER	*pk_ver	ER2/R2	Start address of the packet where version information is to be returned

Return Parameters:

T_VER	*pk_ver	ER2/R2	Start address of the packet where version information is stored
ER	ercd	R0	Error code

Packet Structure:

```
typedef   struct   t_ver {
                   UH maker;      0     2     Manufacturer
                   UH id;         +2    2     Identification number
                   UH spver;      +4    2     Specification version
                   UH prver;      +6    2     Product version
                   UH prno [4];   +8    8     Product management information
                   UH cpu;        +16   2     CPU information
                   UH var;        +18   2     Variation descriptor
                   } T_VER;
```

Error Codes:

E_OK	H'0000	[k]	Normal termination
E_PAR	H'ffdf (–H'21)	[p]	Invalid address (pk_ver is 0 or an odd address)

Description:

The system call get_ver reads information on the version of the kernel currently in use and returns it to the 20-byte RAM area indicated by pk_ver.

The following information is returned to the packet indicated by pk_ver.

(maker)

maker: Manufacturer of this product

The HI2000/3 maker value is H'000a.

(id)

id: Number to identify the OS or VLSI type

The HI2000/3 id value is H'0005.

(spver)

Number to identify the TRON specification series

μITRON specifications: H'5

Version number of the TRON specifications which the product is based

Ver 3.02: H'302

The HI2000/3 spver value is H'5302.

(prver)

This indicates the version number.

Ver 1.0: H'0100

The HI2000/3 prver value is H'0100.

(prno)

This indicates product management information and the product number.

The HI2000/3 prno values are all H'0000.

(cpu)

Same value as that indicated by (maker)

Hitachi, Ltd.: H'0a

The processor executing the OS based on the μITRON specifications

H8S/2600: H'26

H8S/2000: H'20

The HI2000/3 cpu value is H'0a26 or H'0a20.

(var)

Variation descriptor var shows the following:

Kernel specification levels

μITRON level R specifications: B'0100

Reserved. Always read as B'0.

Single processor: B'0

Virtual memory support

Not supported: B'0

MMU support

Supported: B'0

Reserved. Always read as B'0.

File specification level

Not supported: B'000

Reserved. Always read as B'0000.

The HI2000/3 var value is H'4000.

Section 4 Debugging Extension

4.1 Overview

The HI2000/3 Debugging Extension (DX) (hereinafter referred to as HI2000/3 DX) is used by installing it in the Hitachi Debugging Interface (HDI) and the HI2000/3 system.

4.1.1 Displaying and Manipulating Objects

Select a window from the HDI [View] menu to display and manipulate the HI2000/3 DX objects. Table 4.1 lists the menu items added to the HDI [View] menu by the HI2000/3 DX.

Table 4.1 Menu Items Added to the HDI [View] Menu

View Menu	Status Bar
Task List	Open Task List
Trace System	Open System Trace
Event Flags	Open Event Flag
Variable Memory Pool	Open Variable Memory
Fixed Memory Pool	Open Fixed Memory
Semaphores	Open Semaphore
Mailboxes	Open Mailbox
Cyclic Handler	Open Cyclic Handler

Selecting a window in the HDI [View] menu displays the object state.

There are two types of windows: List-type windows, which are generally used, and hierarchical-type windows. An example of a list-type window is shown in figure 4.1, and an example of a hierarchical-type window is shown in figure 4.2.

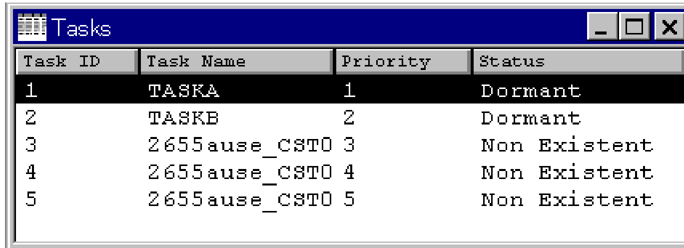


Figure 4.1 Example of the Display of an Object (List-Type Window)

Figure 4.1 shows the Tasks window which displays the Task ID, Task Name, Priority (current task priority), and Status (current task state).

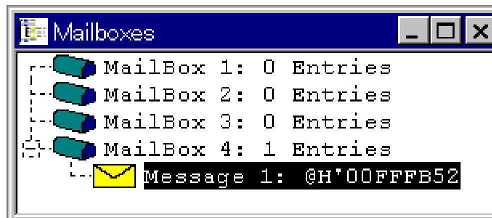


Figure 4.2 Example of the Display of an Object (Hierarchical-Type Window)

Figure 4.2 shows the Mailboxes window which displays all states of the mailbox (task ID at the head of the wait queue or the number of messages). This window can also display the message queue state hierarchically or the message addresses.

A request to manipulate an object can be made from an object window through a dialog box, as shown in figure 4.3. To open a dialog box, first open the pop-up menu by clicking the right button of the mouse in a window and select the menu to open a dialog box. The HI2000/3 DX can only make requests to the kernel through the debug daemon; only the kernel can manipulate objects.

Figure 4.3 shows an example of object manipulation display.

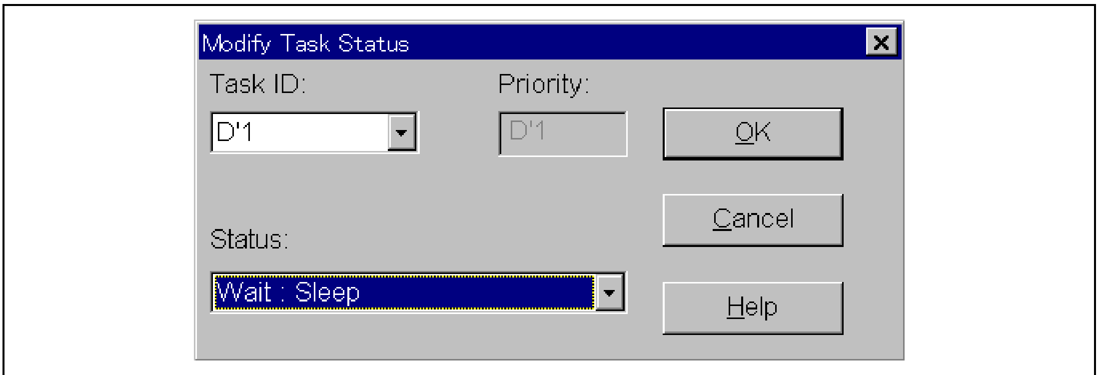


Figure 4.3 Example of Requesting Object Manipulation

Figure 4.3 is called the Modify Task Status dialog box and can modify the task state. The Task ID or status can be modified by the drop down list of the Task ID and Status combo box.

4.1.2 Results of Object Manipulation

The results of object manipulation are shown as the object state in each window.

Each window is updated in the following cases:

- When [Update] is selected from the pop-up menu by clicking the right button of the mouse in each window.
- When the user system stops (at a breakpoint or due to other causes).

4.1.3 Displaying the Register Values

Register values of a task can be displayed by selecting the [View Context] option in the task list pop-up menu. Figure 4.4 shows an example of the register value display.

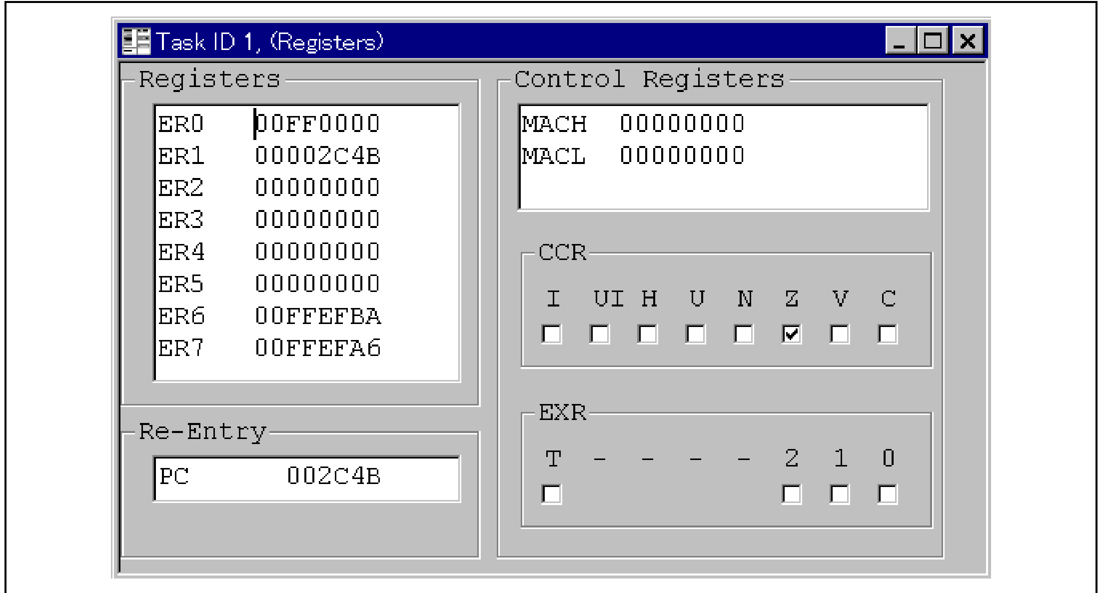


Figure 4.4 [Task Context Registers] Window

Figure 4.4 shows the Task Context Registers window which can be used to edit the task register values while the program is stopped.

4.1.4 Displaying the HI2000/3 DX System Call Trace Information

All trace information concerning the executed system calls can be displayed by acquiring the information from the kernel trace buffer (figure 4.5). The latest information is displayed in the line cycle: –0.

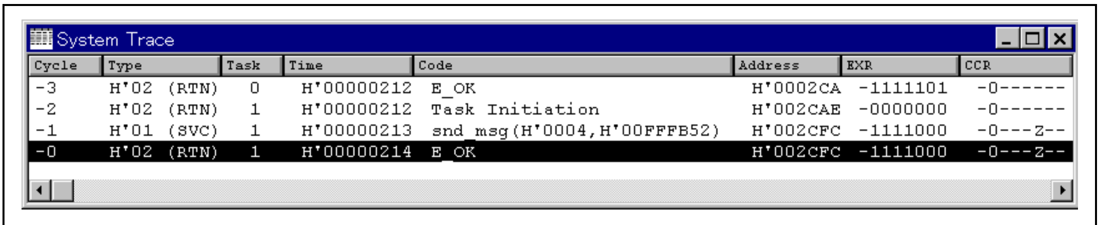


Figure 4.5 [System Trace] Window

Figure 4.5 shows the System Trace window which can display the system calls issued from tasks and the return values as trace information.

4.1.5 Online Help

Context-sensitive help system is available for the standard Microsoft® Windows® operating system. Refer to the online help for details on the HI2000/3 DX operation, windows, and dialog boxes. To open the online help, either press the [F1] key when the HI2000/3 DX window is active, or click the [Help] button in a dialog box.

4.2 List of Functions

4.2.1 HI2000/3 DX Menus

Table 4.2 shows the HI2000/3 DX menus.

Table 4.2 HI2000/3 DX Menus

Menu Bar	Pull-down Menu	Function
View	Task List	Opens [Tasks] window
	Trace System	Opens [System Trace] window
	Event Flags	Opens [Event Flags] window
	Variable Memory Pool	Opens [Variable Length Memory Pool] window
	Fixed Memory Pool	Opens [Fixed Length Memory Pool] window
	Semaphores	Opens [Semaphore] window
	Mailboxes	Opens [Mailboxes] window
	Cyclic Handler	Opens [Cyclic Handler] window

4.2.2 Windows and Dialog Boxes

Table 4.3 shows the list of windows and dialog boxes. Refer to the online help for details. To open the online help, press the [F1] key when the HI2000/3 DX window is active or click the [Help] button in the dialog box.

Table 4.3 HI2000/3 DX Windows and Dialog Boxes

Classification	Window and Dialog Box	Function
Task	[Tasks] window	Displays the state of all tasks
	[Task Modification] dialog box	Modifies task state
Event flag	[Event Flag] window	Displays the state of all event flags
	[Event Flag Modification] dialog box	Modifies event flag state
Semaphore	[Semaphore] window	Displays or modifies the state of all semaphores
Mailbox	[Mailboxes] window	Displays the state of all mailboxes
	[Mailbox Post message] dialog box	Sends messages to mailboxes
Fixed-length memory pool	[Fixed Length Memory Pool] window	Displays the state of all fixed-length memory pools
Variable-length memory pool	[Variable Length Memory Pool] window	Displays the state of all variable-length memory pools
Timer	[Timer] window	Displays the system clock value
	[Time Modification] dialog box	Modifies the system clock value
Trace	[System Trace] window	Displays trace information
Task context	[Task Context Local Variables] window	Displays the task-context local variables
	[Task Context Registers] window	Displays the task-context register values
	[Edit Value] dialog box	Modifies the task-context local variables
	[Registers] dialog box	Modifies the task-context register values
Cyclic handler	[Cyclic Handler] window	Displays the state of the cyclic handler
	[Activate Cyclic] dialog box	Modifies the state of the cyclic handler (active or non-active)

4.3 Notes

4.3.1 Setting up the E6000 Emulator

To update a window during program execution, set up the E6000 as follows:

1. Display the configuration dialog box by selecting [Configure Platform...] from the [Setup] menu.
2. Select the [Enable read and write on the fly] check box in the configuration dialog box so that memory can be accessed during program execution.

4.3.2 Displaying the HI2000/3 DX Window

When displaying the HI2000/3 DX window for the first time after HDI initiation, it may take about one minute since information required to display the HI2000/3 DX window must be acquired from the user system.

4.3.3 Realtime Operation of the User System

The HI2000/3 DX operates by referring to or updating the memory of the user system. If the following functions are performed during the user system operation, the memory will be accessed, and as a result, the user system will not operate in realtime.

- When the HI2000/3 DX window has been opened or updated
- When the [OK] button has been clicked in the dialog box

Since the debug daemon operates cyclically in the user system, the throughput of the user system decreases slightly (less than when performing the above items) even when the HI2000/3 DX functions are not used.

4.3.4 Displaying Correct Data

The HI2000/3 DX directly reads the memory contents of the user system when referring to the object status. Therefore, correct information may not be displayed in the following cases:

- Display during program execution
  When the memory is read while the kernel (program) is being executed
- Display before HI2000/3 initiation
  When the HI2000/3 initiation is not completed before system initialization handler is initiated

4.3.5 Trace

A trace function to trace HI2000/3 system calls must be defined to display the HI2000/3 DX [System Trace] window. For details on defining a trace function, refer to section 6.2.6, Defining Trace Functions.

4.3.6 User System Memory

The memory size used at HI2000/3 DX shipment is shown in table 4.4. The kernel memory size (ROM area) may increase at a maximum of 2.9 bytes by the system calls linked by the debug daemon.

Table 4.4 Memory Size Used by the User System

Memory Type	Used Memory Size
ROM area (only the debug daemon)	Maximum of 500 bytes
RAM area	Maximum of 200 bytes

4.3.7 Correspondence to the HDI Session

The HI2000/3 DX does not correspond to the HDI session. The HI2000/3 DX setting is not stored even when the session is stored.

4.3.8 Loading Load Modules

Keep the HI2000/3 DX windows open when loading load modules after using the HI2000/3 DX.

4.4 Debug Daemon

Installing the Debug Daemon:

The debug daemon must be installed in the system to use the HI2000/3 DX. The debug daemon operates as a cyclic handler.

To install the debug daemon, specify -define=DX="Action" as an assembly option when assembling the CPU initialization routine (***nnnnz***cpu.src) or the setup table (***nnnnz***sup.src).

1. Modifying the Setup table

   The cyclic time interval of the debug daemon can be obtained by the following formula. The cyclic time interval must be changed according to the user system.

   Debug daemon cycle = cyclic time interval x hardware timer cycle

   The cyclic time interval of the debug daemon must be approximately 50 ms. For the provided timer driver, the hardware timer cycle is 10 ms, and the cyclic time interval of the debug daemon is set as 5 ms in the setup table.

   For details on modifying the setup table, refer to section 6.2.5, Defining Cyclic Handlers.

2. Modifying the CPU Initialization Routine

   If the user has created a new CPU initialization routine and is not using the provided one, the debug daemon initial processing must be added to the new one. The following shows how to modify the new CPU initialization routine.

   — Import _HI_DEAMON_IHI

   — Add subroutine call instruction (jsr @_HI_DEAMON_INI) to the CPU initialization routine before jumping to the kernel initial processing (jmp @_H_2S_INIT)

The sample setup table (***nnnnz***sup.src) and the sample CPU initialization routine (***nnnnz***cpu.src) have already been modified.

4.5 Tutorial

This section explains the operation of the HI2000/3 DX using the sample program.

Before starting the tutorial program, a load module must be created. The provided sample files have already been modified as shown in Installing the Debug Daemon in section 4.4. Therefore, a load module shown in the sample program can be created by configuring a system by adding DX=Action to the define option of the HEW workspace project. Refer to section 8, Load Module Creation and create a load module.

The sample program assumes the use of the H8S/2655 advanced mode. Use the debugging extension by replacing the H8S/2655 mode with one suitable for the user environment.

In the sample program, there are two tasks: TASKA and TASKB. TASKA sends a message to mailbox 4 (issues system call snd_msg), receives a message from mailbox 3 (issues system call rcv_msg), and enters sleep state. TASKB receives a message from mailbox 4 (issues system call rcv_msg), sends a message to mailbox 3 (issues system call snd_msg), and enters sleep state.

Figure 4.6 shows the sample processing flow.

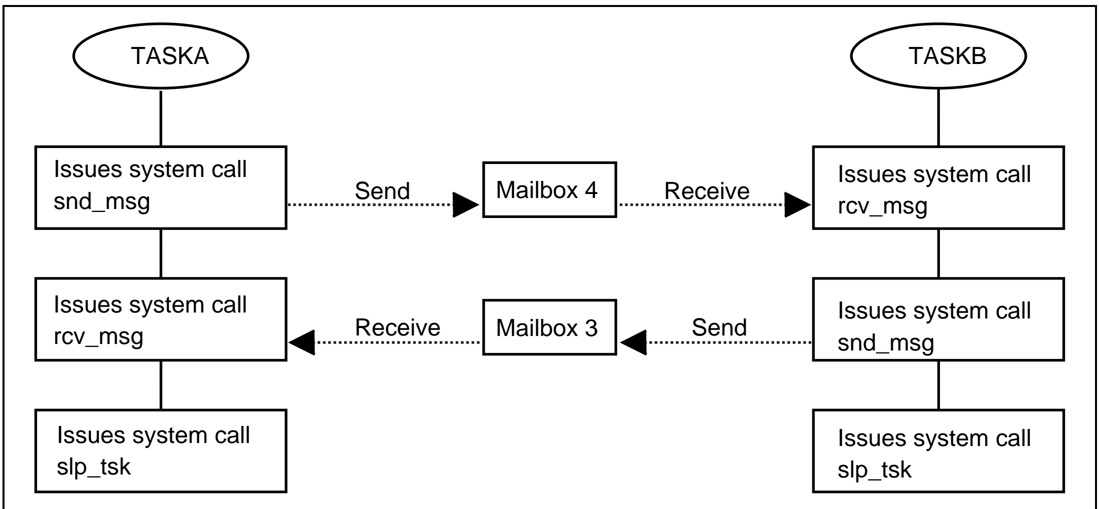


Figure 4.6 Sample Program Processing

This example uses the HDI. For details on the HDI operation, refer to the Hitachi Debugging Interface User's Manual or the E6000 Emulator User's Manual of the H8S series microcomputer used.

4.5.1 Executing a Sample Program

1. Invoke the HDI by selecting the [HDI for E6000 H8S] icon. The HDI initial display as shown in figure 4.7 appears.

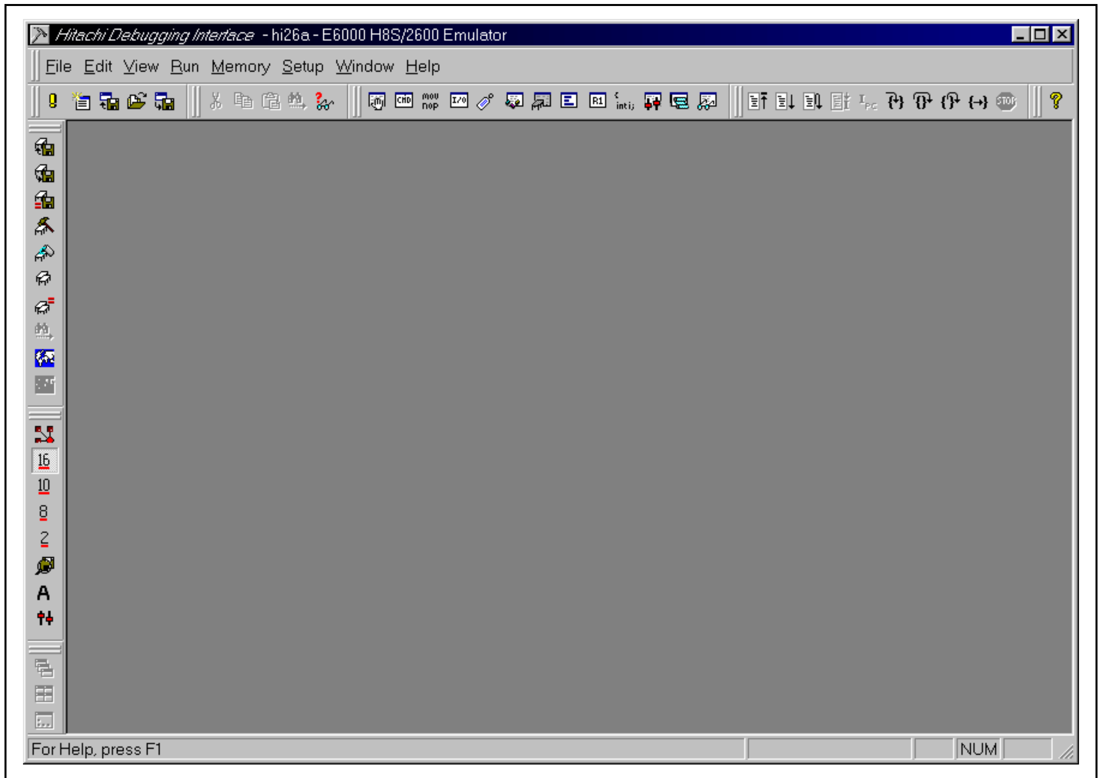


Figure 4.7 HDI Initial Display

2. Select [Load Program…] from the [File] menu and open the [Open] dialog box. Then load application load module `hi26a.abs` under directory hi26a to the emulator read/write area. Here, the program does not operate since the kernel is not initiated.

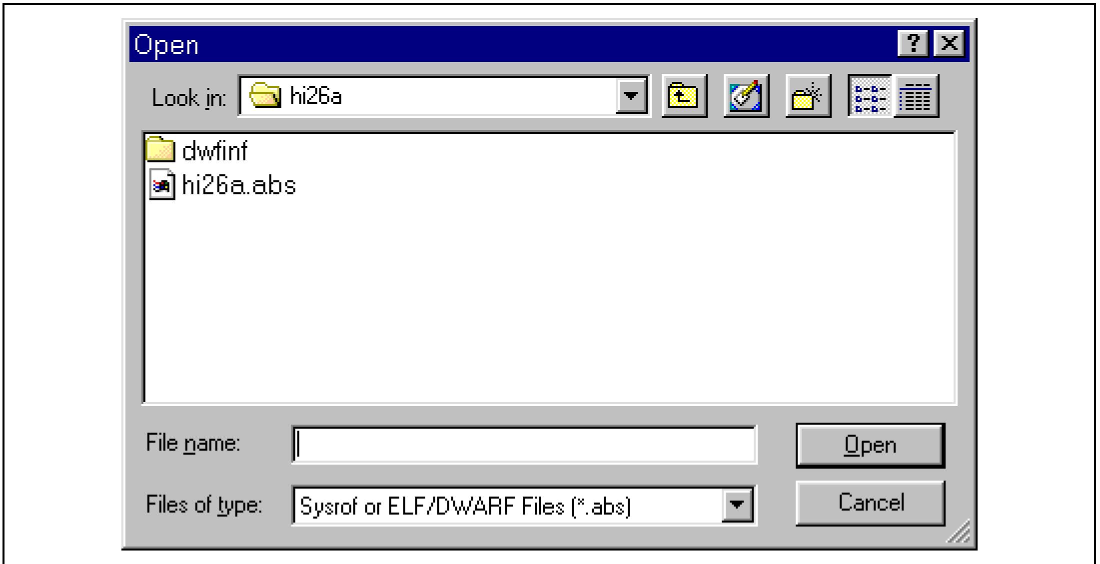
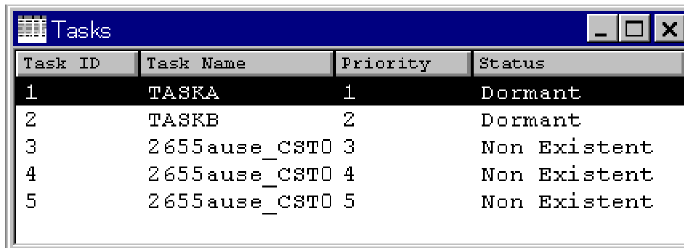


Figure 4.8 [Open] Dialog Box

3. Execute the program by selecting [Go Reset] from the [Run] menu. Pressing the STOP button after a few seconds initializes the system and opens the HDI source window.
4. Open the [Tasks] window by selecting [Task List] from the [View] menu to check the task state. The task is executed according to the task definition table in the `2655asup.src`. The first two tasks are in the DORMANT state and the next three tasks are in the NON-EXISTENT state.


Tasks

Task ID	Task Name	Priority	Status
1	TASKA	1	Dormant
2	TASKB	2	Dormant
3	2655ause_CST0	3	Non Existent
4	2655ause_CST0	4	Non Existent
5	2655ause_CST0	5	Non Existent

Figure 4.9 [Tasks] Window

4.5.2 Starting a Task

1. Set a breakpoint before executing the program. Open the [Open] dialog box by selecting [Source...] from the [View] menu and open file `task.c`. The source code window of the two DORMANT tasks, TASKA and TASKB, appears. Set a breakpoint at the snd_msg line of TASKA.

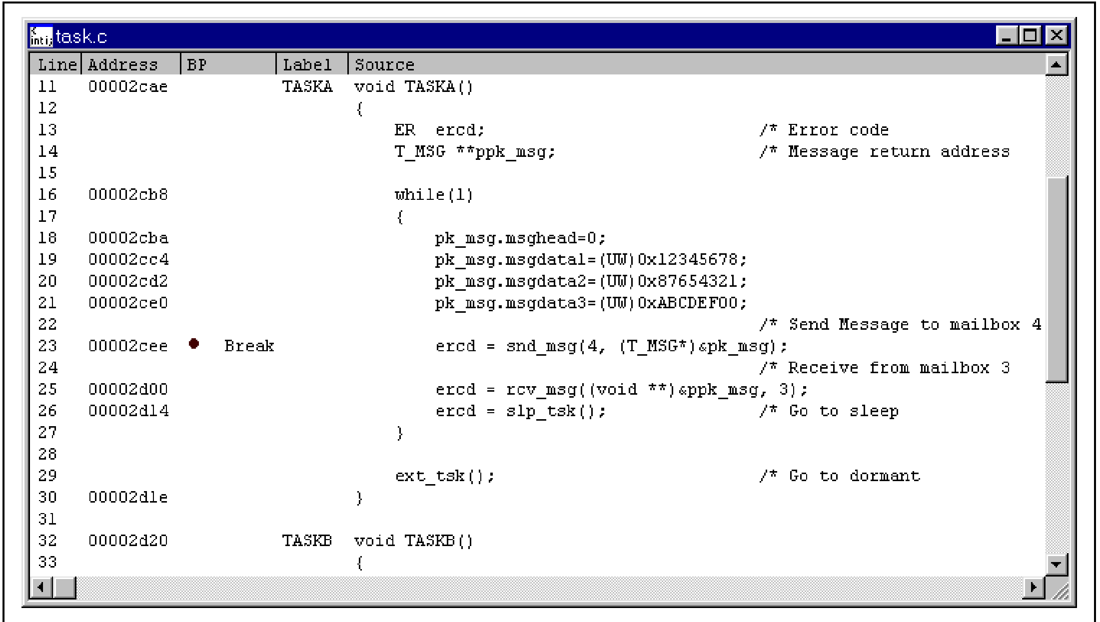


Figure 4.10 Source Code Display

2. Initiate TASKA (task ID1) by opening the [Modify Task Status] dialog box (figure 4.11). The display does not change until the program is executed. Execute the program by selecting [Go] from the [Run] menu. The program stops when it reaches a breakpoint.

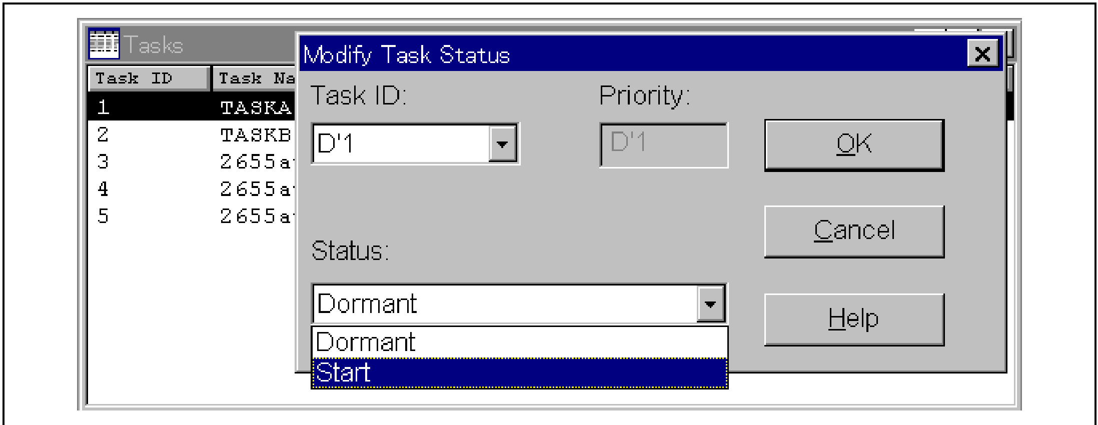


Figure 4.11 Invoking Task

3. Check the operation of step 2. Open the [System Trace] window by selecting [Trace System] window (figure 4.12) from the [View] menu. The system trace information of up to four entries before termination is displayed, where cycle: –0 is the latest trace information. Table 4.5 describes the trace contents.

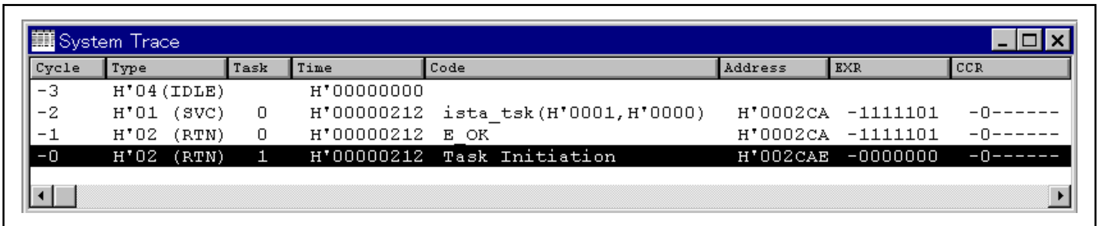


Figure 4.12 [System Trace] Window

Table 4.5 Description of Trace Contents

Cycle	Description
–3	System is in the idle state before TASKA (task ID1) starts.
–2	ista_tsk is called from the task-independent portion.
–1	The error code of the system call ista_tsk.
–0	Shows the return attribute for TASKA (task ID1), indicating that the task has started.

4.5.3 Mailboxes and Messages

1. Check mailbox state by selecting [Mailboxes] from the [View] menu and opening the [Mailboxes] window (figure 4.13).

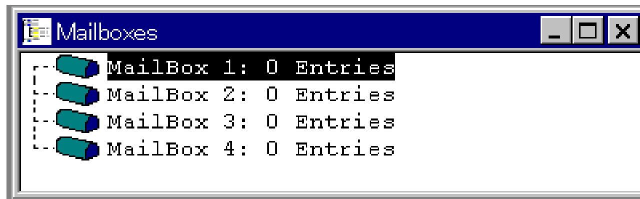


Figure 4.13 [Mailboxes] Window

2. Send a message to mailbox 4. Open the [Open] dialog box by selecting [Source...] from the [View] menu and open file `task.c`. The source code window for the two DORMANT tasks, TASKA and TASKB, is displayed. Step over the ercd=snd_msg(4,(T_MSG*)&pk_msg) line of TASKA by selecting [Step Over] from the [Run] menu (figure 4.14).

Figure 4.14 Step-Over Execution of Program

3. As a result, one entry appears in mailbox 4.

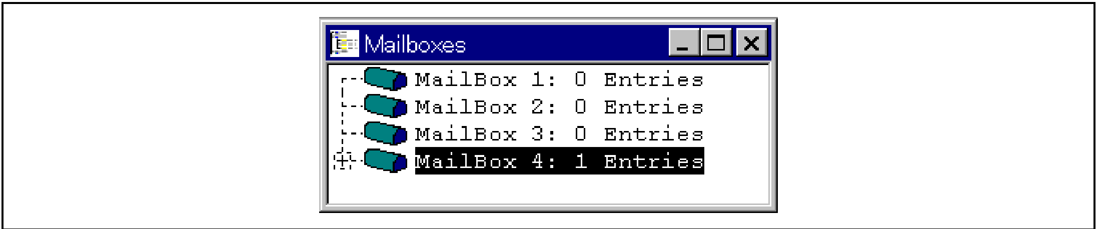


Figure 4.15 [Mailboxes] Window (Confirmation of Result)

Click + with the right button of the mouse to expand this mailbox. The start address H'00FFFB52 of a message is displayed.

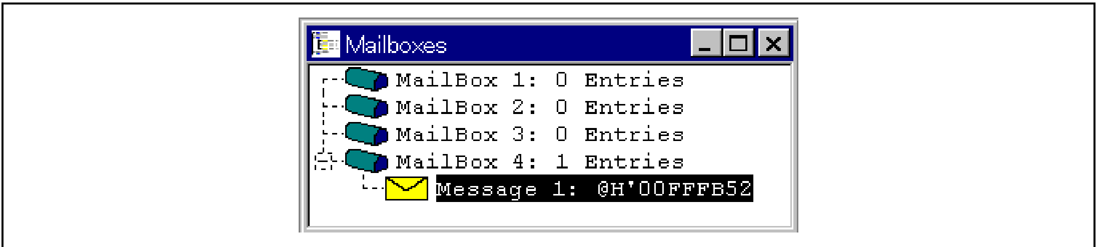


Figure 4.16 [Mailboxes] Window (Expanded Display)

4. To check program operation, open [System Trace] window by selecting [Trace System] from the [View] menu (figure 4.17). In the window, trace information is displayed, indicating that a message is sent (system call snd_msg is issued) at cycle –1, and a system call response (return value) is received at cycle –0.

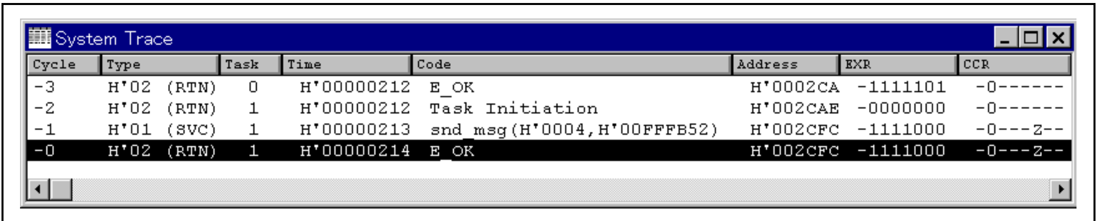


Figure 4.17 [System Trace] Window

4.5.4 Examples during System Operation

Sections 4.2.1 to 4.2.3 shows examples when the system was not operating. This section describes examples during system operation.

1. To display [Tasks] and [Mailboxes] windows, open the [Tasks] window by selecting [Task List] from the [View] menu. Similarly, open the [Mailboxes] window by selecting [Mailboxes] from the [View] menu.
2. To execute the program, select [Go] from the [Run] menu.
3. To check task state, select the [Update] option from the [Task List] window pop-up menu. TASKA (task ID1) is in the WAIT state waiting for a message from mailbox 3 (figure 4.18).

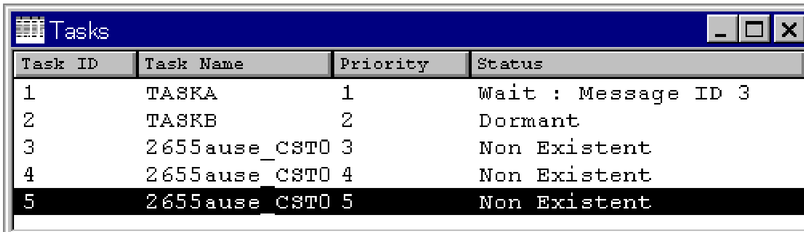


Figure 4.18 [Tasks] Window after the [Update] Option is Selected

4. To check the mailbox state, select the [Update] option from the [Mailboxes] window pop-up menu. TASKA (task ID1) is the wait task for mailbox 3.

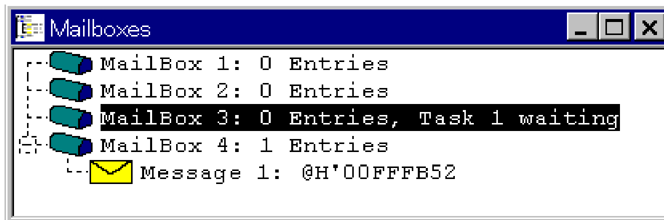


Figure 4.19 [Mailboxes] Window after the [Update] Option is Selected

5. To check system trace results, select [Trace System] from the [View] menu and open the [System Trace] window. The display shows that TASKA (task ID1) called rcv_msg and the system entered the idle state.

System Trace

Cycle	Type	Task	Time	Code	Address	EXR	CCR
-3	H'01 (SVC)	1	H'00000213	snd_msg(H'0004,H'00FFFB52)	H'002CFC	-1111000	-0---Z--
-2	H'02 (RTN)	1	H'00000214	E_OK	H'002CFC	-1111000	-0---Z--
-1	H'01 (SVC)	1	H'00000215	rcv_msg(H'0003)	H'002D10	-1111000	-0---Z--
-0	H'04(IDLE)		H'00000215				

Figure 4.20 [System Trace] Window

6. Invoke TASKB (task ID2) and send a message to mailbox 3. Select TASKB (task ID2) in the [Task List] window, and select [Edit Properties] from the pop-up menu. The [Modify Task Status] dialog box appears (figure 4.21). Select [Start] from the [Status] box, and invoke the task. TASKB (task ID2) will send a message to mailbox 3.

   In the sample program, when TASKB (task ID2) sends a message to mailbox 3, TASKA (task ID1) receives it, and both tasks enters sleep state.

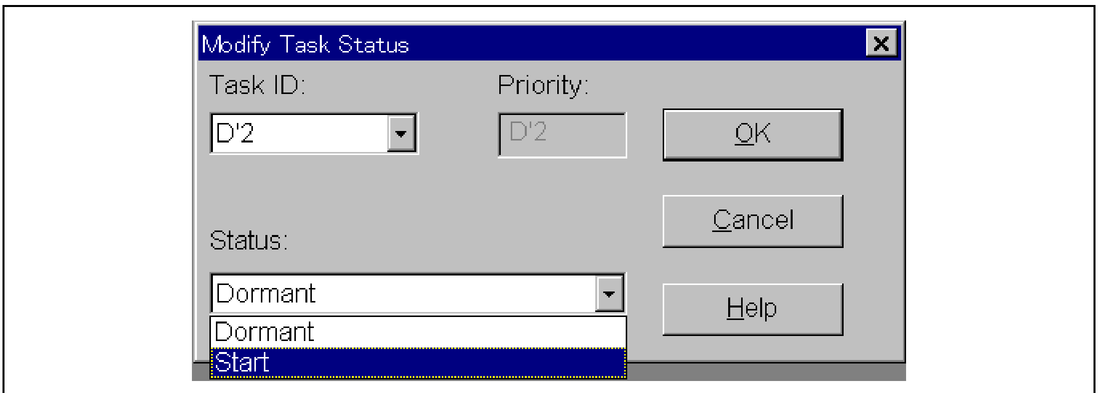


Figure 4.21 [Modify Task Status] Dialog Box

7. To check the results, select the [Update] option from the [Task List] window pop-up menu, and update the window contents. The display shows that sending and receiving the message is completed and that TASKA (task ID1) and TASKB (task ID2) entered sleep state (figure 4.22).

Tasks

Task ID	Task Name	Priority	Status
1	TASKA	1	Wait : Sleep
2	TASKB	2	Wait : Sleep
3	2655ause_CST0	3	Non Existent
4	2655ause_CST0	4	Non Existent
5	2655ause_CST0	5	Non Existent

Figure 4.22 [Tasks] Window after the [Update] Option is Selected

In the [System Trace] window, system call slp_tsk issued to TASKA and TASKB and the system entering the idling state is displayed (figure 4.23).

System Trace

Cycle	Type	Task	Time	Code	Address	EXR	CCR
-3	H'01 (SVC)	1	H'00000753	slp_tsk()	H'002D18	-1111000	-0---Z--
-2	H'02 (RTN)	2	H'00000753	E_OK	H'002D82	-1111000	-0---Z--
-1	H'01 (SVC)	2	H'00000753	slp_tsk()	H'002D8A	-1111000	-0---Z--
-0	H'04(IDLE)		H'00000753				

Figure 4.23 [System Trace] Window

Section 5 Creating Application Programs

5.1 Creating a User Program

User programs can be written in C language or an assembly language.

The following show the programs created.

- Tasks: programs divided into units that run independently and in parallel
- Handlers and Routines: programs that are invoked when interrupts occur (CPU initialization routine, system termination routine, timer initialization routine, system initialization handler, system idling routine)

Programs must be created according to the user system. Figure 5.1 shows an outline of kernel initial processing.

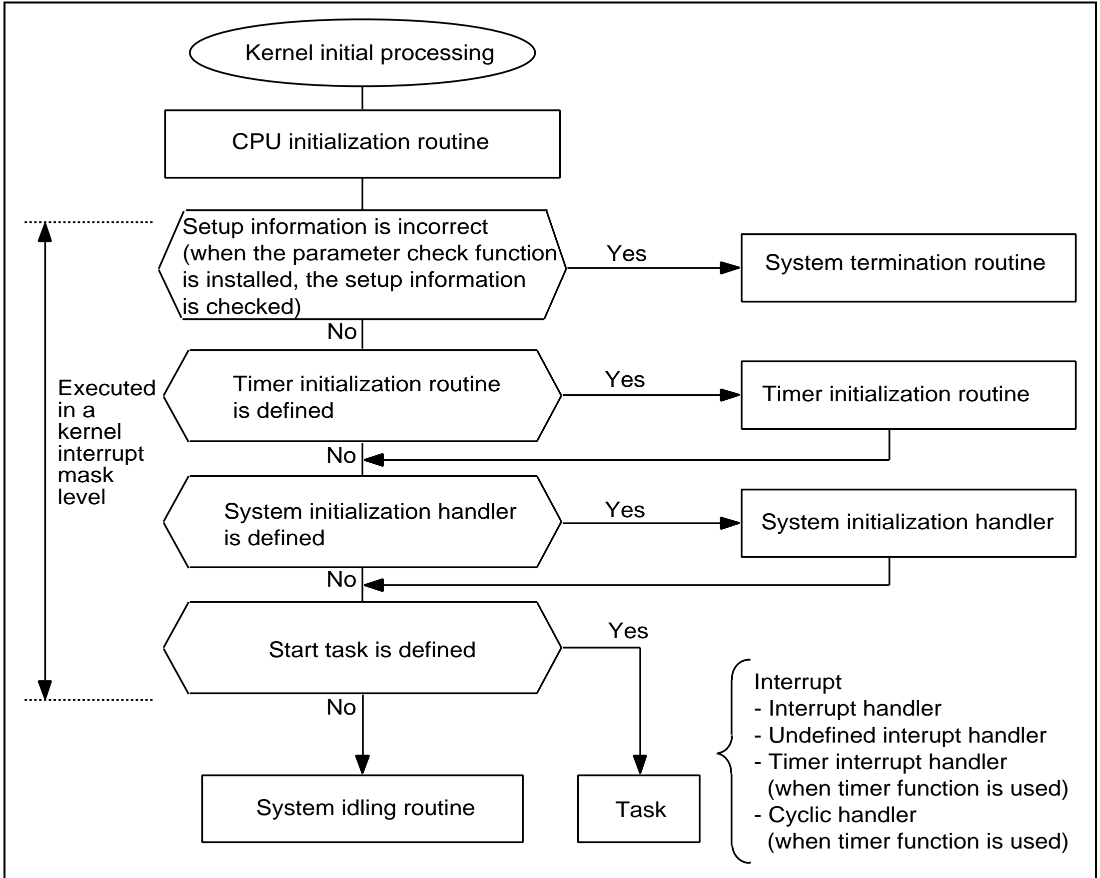


Figure 5.1 Kernel Initial Processing

5.2 Tasks

5.2.1 Creating Tasks

Figure 5.2 shows an example of a task. A task must be terminated by issuing system call ext_tsk at the end of the task. If the operation returns from the task start function to the caller, normal system operation cannot be guaranteed.

Programs written in C language can be used in normal mode or advanced mode according to the CPU option or environment variable specification .

```
#include "hi2000.h"


void    task(INT stacd)
INT stacd;
{
ID      tskid;                    /* task ID                    */
ER      ercd;                     /* error code                 */


        .............
        .............

        ercd = wup_tsk(tskid);    /* system calls that can be      */
                                  /* issued from the task portion  */


        .............
        .............

        ext_tsk();                /* system calls that can be      */
                                  /* issued from the task portion  */
}
```

Figures 5.2 Task Example in C Language

When a task execution request (event) is issued, the kernel will control the execution of tasks based on their state in the system and on the priority assigned to them and performs task processing. A priority with a smaller value indicates a higher priority.

1. Resources Initialized at Task Initiation

   When a task is initiated, resources related to the task are initialized, as shown in table 5.1. System call sta_tsk or ista_tsk is used for task initiation.

Table 5.1 Resources Initialized at Task Initiation

Item	Initialization Specification
Program counter (PC)	The task start address specified at the task definition
Condition code register (CCR)	Interrupt mask cancellation (0)
Extend register (EXR)	Interrupt mask cancellation (0)
Stack pointer (ER7)	Task stack pointer specified at the task definition
R0 (written in assembly language)/ first parameter (written in C language)	A random task initiation code (stacd) specified by the sta_tsk system call
General register (E0, ER1 to ER6) and multiply and accumulate register (MACH and MACL)	Undefined (the multiply and accumulate registers (MACH and MACL) are for the 2600 CPU)
Task priority	The initial task priority specified at the task definition
Task wakeup request (wupcnt)	0
Task suspend request	Cannot be nested

2. Pre-Termination Processing

   Resources acquired with the system calls must be returned before a task enters the DORMANT state. Resources and related system calls are listed in table 5.2. System call ext_tsk or ter_tsk is used to place a task in the DORMANT state.

Table 5.2 Resources and System Calls

Resource	Specification	System Calls to Acquire Resource	System Calls to Return Resource
Semaphore count	Number of resources acquired by P instruction	wai_sem, preq_sem, or twai_sem	sig_sem
Memory block	Acquired from the fixed-size memory pool	get_blf, pget_blf, or tget_blf	rel_blf
	Acquired from the variable-size memory pool	get_blk, pget_blk, or tget_blk	rel_blk

3. Monopolizing the CPU and Masking Interrupts during Task Execution

   a. Monopolizing the CPU by issuing the loc_cpu system call

      To monopolize the CPU during task execution, lock the CPU by using the loc_cpu system call. Unlock the CPU by issuing the unl_cpu system call to execute tasks again. The CPU-locked state is different from the task-execution state in the following respects.

      - System calls dedicated to task portion

        System calls that shift a task into the WAIT state cannot be issued.

      - System calls dedicated to the task-independent portion

        System calls dedicated to the task-independent portion cannot be issued.

      - Task switch delay

        Even if task switching becomes necessary while the CPU is being locked, it will be delayed until the CPU is unlocked by issuing a unl_cpu system call.

      - Interrupt masking

        Interrupts having an interrupt level equal to or lower than the kernel interrupt mask level, which is defined in the setup table, are masked.

   b. Monopolizing the CPU by issuing the chg_ims system call

      Masking interrupts during task execution shifts the execution from task portion to the task-independent portion. Use system call chg_ims to change the interrupt mask level. If interrupts are masked during task execution, task dispatch will not occur while interrupts are masked and the task can monopolize the CPU. Since the system is placed in the task-independent portion while interrupts are being masked during task execution, it is different from the task-execution state in the following respects.

      - System calls dedicated to the task portion

        System calls dedicated to the task portion cannot be issued. If the kernel library has a parameter check function, error code E_CTX is returned. If the kernel library does not have a parameter check function, normal system operation cannot be guaranteed.

      - System calls dedicated to the task-independent portion

        If system calls dedicated to the task-independent portion are issued, normal system operation cannot be guaranteed.

      - Task switch delay

        Even if task switching becomes necessary while interrupts are being masked, it will be delayed until the interrupt mask level is changed to 0 by issuing the chg_ims system call.

      - Interrupt masking

        Do not issue a system call other than chg_ims when interrupts are masked at a level higher than the kernel interrupt mask level defined in the setup table.

5.2.2 Defining Tasks

Define the task start address, initial priority, task stack size, task initial state, and extended information at task definition. For details on task definition, refer to section 6.2.2, Defining Task.

For details on defining tasks, refer to section 6.2.2, Defining Task.

5.3 Interrupt Handlers

5.3.1 Interrupt Handler Description

When an interrupt occurs, control is passed to the interrupt handler without kernel intervention. Therefore, the interrupt handler must store the register contents when an interrupt occurs, and restore them when it has finished. The interrupt handler operates as follows.

1. The contents of the registers to be used by the interrupt handler are stored
   a. The stack pointer value is saved
   b. The stack pointer is moved to the interrupt-handler stack area (not necessary if the interrupt handler does not use the stack)
   c. The register contents are pushed onto the stack
2. The interrupt is processed
3. The contents of the registers used by the interrupt handler are restored
   a. The register contents are restored from the stack
   b. The stack pointer is restored (not necessary when the interrupt handler does not use the stack)
4. A ret_int system call is issued (when the interrupt mask level of the interrupt handler is equal to or lower than that of the kernel) or the RTE instruction is executed (when the interrupt mask level of the interrupt handler is higher than that of the kernel)

An interrupt function can be written in C language by using interrupt function (#pragma interrupt) of the H8S, H8/300 series C compiler.

The #pragma interrupt directive declares the function to be used as an interrupt handler. In this example, the inthdrxx function is declared as an interrupt handler. The stack switching and interrupt function termination are specified.

The stack switching specifies the stack area to be used for interrupt handler processing. The initialization stack pointer is specified by sp = <address>. In this case, a stack area dedicated to each interrupt level must be specified.

The interrupt function termination specification specifies how to return from the interrupt handler. In the interrupt function termination specification system call ret_int or instruction RTE must be executed at the end of the handler. If the interrupt handler has a level equal to or lower than the kernel interrupt mask level, write sy = $ret_int to execute the jmp @ret_int instruction at the end

of the handler, which will call the ret_int system call. If the interrupt handler has a level higher than the kernel interrupt mask level, nothing needs to be written.

For details, refer to the H8S, H8/300 Series C/C++ Compiler User's Manual. Programs written in C language can be used in normal mode or advanced mode according to the CPU option or environment variable specification.

Figure 5.3 shows an example of an interrupt handler.

```
#include "hi2000.h"

extern        VH  hi_intstkxx[ ];
static const  VP  P_intstkxx  =  (VP)&hi_intstkxx[60];

#pragma interrupt (inthdrxx(sp=P_intstkxx, sy=$ret_int))
void inthdrxx( void )         /* The data type of the interrupt handler*/
                              /* function is void                      */
{
ID     tskid;                 /* Task ID                               */
ER     ercd;                  /* Error code                            */
UINT   imask;
                :
                :
     ercd = chg_ims(imask);   /* System calls dedicated to task-       */
                              /* independent portion                   */
                :
                :
     ercd = iwup_tsk(tskid);  /* System calls dedicated to task-       */
                              /* independent portion                   */
                :
                :
}
```

Figure 5.3 Interrupt Handler Example in C Language

Table 5.3 lists the conditions for interrupt handler processing.

Table 5.3 Conditions for Interrupt Handler Processing

Item	Description
Interrupt mask	The interrupt handler is initiated when interrupts are masked at the specified interrupt mask level.
Usable registers	ER0 to ER6, MACH, and MACL (MACH and MACL are for the 2600 CPU only) can be used. Must be saved in stacks at the interrupt handler initiation and later restored at handler termination.
Stack pointer	Set to the same value as that at initiation when control is returned to the point where the interrupt handler is initiated.
Usable system calls	System calls dedicated to task-independent portion. No system call can be issued from the NMI interrupt handler or an interrupt handler whose interrupt mask level is higher than that of the kernel.
Usable stack area	When using the stack, reserve a stack area for the interrupt handler at system configuration and switch the stack at interrupt handler initiation. Interrupt handlers having the same interrupt level can share a stack area.
Termination	Execution is terminated by the ret_int system call. Restore the stack value at termination. Use the RTE instruction to terminate an NMI interrupt handler or an interrupt handler whose interrupt mask level is higher than that of the kernel.

When creating interrupt handlers, keep in mind the following precautions.

1. Guarantee of interrupt mask levels

   The kernel supports all four interrupt control modes.

   The interrupt control bits of the CCR register and EXR register determine the interrupt mask level.

   In interrupt handlers, the processing for an interrupt mask level differs depending on the interrupt control mode.

   a. Interrupt control mode 0

      In this mode, the kernel controls the interrupt mask level using only the I bit of the CCR register. When the interrupt handler is initiated, the I bit of the CCR register is set to 1. Do not clear the I bit in the interrupt handler; otherwise, the kernel system operation cannot be guaranteed.

b. Interrupt control mode 1

In this mode, the kernel controls the interrupt mask level using the I and UI bits of the CCR register. When the interrupt handler is initiated, the I and UI bits of the CCR register are set to 1.

For the interrupt handler with control level 0, clear the UI bit and change the interrupt mask level so that interrupts with control level 1 can be accepted. Do not clear the I bit of the CCR register for the interrupt handler with control level 1; otherwise, the kernel system operation cannot be guaranteed. The interrupt mask levels cannot be changed in the interrupt handler which has a control level of 1.

c. Interrupt control mode 2

In this mode, the kernel controls the interrupt mask level using the I0 to I2 bits of the EXR register, where the I and UI bits of the CCR register are ignored. When the interrupt handler is initiated, the I0 to I2 bits of the EXR register are set to the mask level value of the corresponding interrupt.

d. Interrupt control mode 3

In this mode, the kernel controls the interrupt mask level using the I and UI bits of the CCR register and the I0 to I2 bits of the EXR register. When the interrupt handler is initiated, the I and UI bits of the CCR register and I0 to I2 bits of the EXR register are set to 1. For the interrupt handler with control level 0, clear the I and UI bits of the CCR register and change the interrupt mask level so that interrupts with a higher priority can be accepted. Do not change the EXR register in the interrupt handler; otherwise, the kernel system operation cannot be guaranteed.

2. Kernel Interrupt Mask Level

The kernel includes a critical section, where it executes while masking interrupts to prevent generating contradictory internal information. Any interrupt generated during the critical section execution is not accepted until the kernel leaves the critical section. However, only interrupts having an interrupt level higher than the kernel interrupt mask level are immediately accepted even during the critical section execution.

Notes: 1. Interrupt handlers having an interrupt level higher than the kernel interrupt mask level are not allowed to issue a system call. If any system call is issued, normal system operation cannot be guaranteed. Execute the RTE instruction to return from an interrupt handler having an interrupt level higher than the kernel interrupt mask level.

2. If the interrupt control mode is 3 and the kernel interrupt mask level is 7, system calls cannot be issued from an interrupt handler whose control level is 1.

3. Notes on Interrupts

   a. The user can create interrupt handlers with few restrictions, but if their execution time is too long, system throughput may drop and system response time may be degraded.

   b. The kernel interrupt mask level can be determined by defining it in the setup table. If the interrupt handler has an interrupt level higher than the kernel interrupt mask level, or if it is the case of an NMI handler, a system call must not be issued. Do not issue a system call from the interrupt handler having an interrupt level higher than the kernel interrupt mask level; otherwise, normal system operation cannot be guaranteed.

   c. Use the ret_int system call to return from an interrupt handler having an interrupt level equal to or lower than the kernel interrupt mask level. Do not use a system call other than the ret_int system call; otherwise, normal system operation cannot be guaranteed.

5.3.2 Defining Interrupt Handlers

To define interrupt handlers, set the start address of an interrupt handler in the appropriate interrupt vector table. Control can be passed to the interrupt handler without kernel intervention. For details on the cause of interrupts, refer to the refer to the target MCU hardware manual.

Figure 5.4 shows the relationship between the vector table and the interrupt handler.

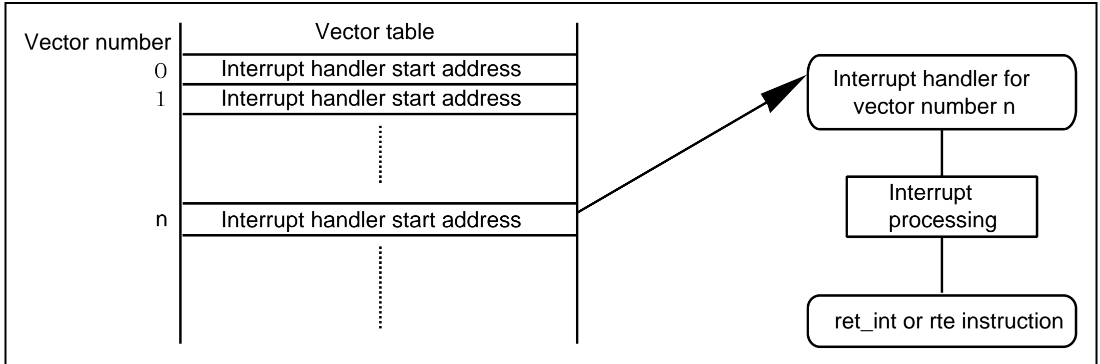


Figure 5.4 Relationship between the Vector Table and the Interrupt Handler

For details on interrupt handler definition, refer to section 7, Creating the Interrupt Vector Table.

5.4 Undefined Interrupt Handlers

5.4.1 Creating Undefined Interrupt Handlers

An undefined interrupt handler is a program that is executed when an unexpected interrupt occurs in the system. The provided undefined interrupt handler program will call the kernel undefined interrupt processing (jsrΔ@_H_ilint) as a subroutine and terminate the system. For details on undefined interrupt handlers at system termination, refer to section 5.9, System Termination Routine, and the CPU Hardware Manual.

An undefined interrupt handler can be created in the same way as an interrupt handler.

5.4.2 Defining Undefined Interrupt Handlers

For details on defining undefined interrupt handlers, refer to section 7, Creating the Interrupt Vector Table.

The sample undefined interrupt handler is sample***nnnnz***smp***nnnnz***ili.src.

5.5 Cyclic Handlers

5.5.1 Creating Cyclic Handlers

A cyclic handler is a task-independent portion and initiated at a specific cycle time.

After a specified cycle time has passed, a cyclic handler is initiated by the timer interrupt handler.

A cyclic handler must store the register contents when it is initiated, and restore them when it has finished. The cyclic handler operates as follows.

1. The contents of the registers to be used by the cyclic handler are stored
   a. The register contents are pushed onto the stack (Use the timer interrupt handler stack)
2. The cyclic handler is processed
3. The contents of the registers used by the cyclic handler are restored
   a. The register contents are restored from the stack
4. The RTS instruction is executed

A cyclic handler can be written in C language by using the extended function (#pragma asm) of the H8S, H8/300 series C compiler. Programs written in C language can be used in normal mode or advanced mode according to the CPU option or environment variable specification.

Specify the output of the assembly program using option code=asemode at compilation. To output an assembly program, it is recommended that the user create a cyclic handler C source program in a different file from the other C source program files.

Figure 5.5 shows an example of an cyclic handler.

```
#include  "hi2000.h"
void      cyc_hdr (void)
#pragma asm
          stm.l    (er0-er1),@-sp      ;: Saves er0 and er1 in stack
          bsr      cychdr_main:8       ;: Calls function
;
          ldm.l    @sp+,(er0-er1)      ;: Restores er0 and er1
          rts                          ;: Executes the rts instruction
cychdr_main;
#pragma endasm
{
             /* Cyclic handler processing */
}
```

Figure 5.5 Cyclic Handler Example for C Language

Table 5.4 lists the conditions for cyclic handler processing.

Table 5.4 Conditions for Cyclic Handler Processing

Item	Description
Interrupt mask	Cyclic handlers are initiated when interrupts are masked at the level of the timer interrupt mask level.
Usable registers	ER0 to ER6, MACH, and MACL (MACH and MACL are for the 2600 CPU only) can be used. Must be saved in stacks at the cyclic handler initiation and later restored at the handler termination. Save and restore ER0 and ER1 when writing the cyclic handler in C language as shown in figure 5.5.
Stack pointer	Uses the timer interrupt handler stack. Must be restored at the handler termination.
Usable system calls	System calls dedicated to task-independent portion.
Usable stack area	Add the stack size used by the cyclic handler to the timer interrupt handler stack size during system configuration.
Returning method	Processing must be terminated by the RTS instruction. Restore the stack value at termination.

Cyclic handlers are executed while interrupts are being masked at the timer interrupt mask level. If a specified cycle time has passed for multiple cyclic handlers, they will be executed while interrupts are masked at the timer interrupt mask level. Therefore, the following may occur.

- Delay in the system clock
- The system response may be degraded for interrupts with a level equal to or lower than the timer interrupt mask level

To avoid this,

- Do not specify an extremely short timer interrupt cycle
- Keep the processing of the cyclic handler as short as possible
- Keep the cycle of the cyclic handler as large as possible

If the cycle time of a cyclic handler is 1, and the handler processing time takes more time than the timer cycle time, the cyclic handler will be repeated infinitely, and the system will be hung.

5.5.2 Defining Cyclic Handlers

The kernel controls the cycle time count and the execution of cyclic handlers according to the information specified in the setup table. Cyclic handlers can be defined by defining them in the setup table. To define a cyclic handler, use the example of the cyclic handler definition field of the setup table.

For details on cyclic handler definition, refer to section 6.2.5, Defining Cyclic Handlers.

5.6 CPU Initialization Routine

5.6.1 Creating CPU Initialization Routines

The CPU initialization routine is a program to initialize the CPU before the kernel is initiated. The CPU initialization routine operates as follows.

1. The stack pointer is specified

2. The CPU is initialized

3. The call of debug daemon initial processing (when the debugging extension (DX) is used)

4. Control jumps (jmp @_H_2S_INIT) to the kernel initial processing (_H_2S_INIT)

The sample CPU initialization routine is written in assembly language.

A CPU initialization routine can be written in C language by using the sample programs (include files directory name: 2600) of the H8S, H8/300 series C compiler. Programs written in C language can be used in normal mode or advanced mode according to the CPU option/environment variable specification.

Figure 5.6 shows an example of a CPU initialization routine.

```
#include "2655s.h"                /* Specifies H8S/2655 include file */
void H_2S_INIT(void);             /* Declares kernel initialization
processing */
#ifdef DX
  void HI_DEAMON_INI(void);       /* Declares daemon initialization
processing */
#endif
#pragma stacksize 0x012           /* Declares stack session */
#pragma entry H_2S_CPUINI         /* Declares entry function  */
void H_2S_CPUINI(void)
{
   SYSCR.BIT.INTM = 3;           /* Sets interrupt control mode */
   MSTPCR.BIT.B13 = 1;           /* Clears module stop bit  */
 #ifdef DX
   HI_DEAMON_INI();             /* Calls daemon initialization processing
*/
 #endif
   H_2S_INIT();                  /* Jumps to kernel initialization
processing  */
}
```

Figure 5.6   CPU Initialization Routine Example

Table 5.5 lists the conditions for CPU initialization routine processing.

Table 5.5 Conditions for CPU Initialization Routine Processing

Item	Description
Interrupt mask	After the CPU has been reset, all interrupts including the NMI are masked.
Usable registers	All.
Stack pointer	The stack pointer must be specified at the start instruction of this processing. Example: > mov.l #xx : 32,sp If an NMI interrupt is generated before the stack pointer has been initialized, normal operation cannot be guaranteed.
Usable system calls	Since the kernel is not yet initiated, no system calls can be used.
Usable stack area	Reserve the stack area if necessary at system configuration and specify the stack at system initiation.
Returning method	Processing must be terminated by jumping to the kernel initiation processing. jmp @_H_2S_INIT

5.6.2 Defining CPU Initialization Routines

To define a CPU initialization routine, specify the label H_2S_CPUINI as the start address of the CPU initialization routine. The label must be declared with the export directive.

The CPU initialization routines are defined in the reset vector (vector numbers 0 and 1).

- Vector number 0: Power-on reset
- Vector number 1: Manual reset

Note: Some H8S series microcomputers may not have a manual reset function. In such a case, simply define a power-on reset. For details, refer to the target MCU hardware manual of the H8S series microcomputer used.

For details on CPU initialization routine definition, refer to section 7, Creating the Interrupt Vector Table.

The sample CPU initialization routine is sample***nnnnz***smp***nnnnz***cpu.src.

5.7 Timer Initialization Routine

The timer initialization routine is necessary together with the timer interrupt handler when using the time-management function. The timer driver consists of two modules: a timer initialization routine and a timer interrupt handler. For details, refer to appendix C, Device Driver.

5.8 System Initialization Handlers

5.8.1 Creating System Initialization Handlers

The system initialization handler is a program called from the kernel initialization process.

The system initialization handler can initialize resources and hardware before starting the start task.

The system initialization handler operates as follows.

1. The contents of the registers to be used by the system initialization handler is stored
    a. The register contents are pushed onto the stack (Guarantee the register contents according to the rules on guaranteeing register contents in C language programs (functions).)
    b. The OS stack is used to store the register contents.
2. The system initialization handler is processed
    a. The number of resources managed by the semaphore is initialized
3. The contents of the registers used by the system initialization handler is restored
    a. The register contents are restored from the stack
4. RTS instruction is executed

A system initialization handler can be written in C language by using the C compiler extended function (#pragma asm) of the H8S, H8/300 series C compiler.

Programs written in C language can be used in normal mode or advanced mode according to the CPU option/environment variable specification.

Specify the output of the assembly program using option code=asemode at compilation. To output an assembly program, it is recommended that the user creates a system initialization handler C source program in a different file from the other C source program files.

Figure 5.7 shows an example of a system initialization handler.

```
#include   "hi2000.h"
void    HIPRG_SYSINI(void)       ;: Label name HIPRG_SYSINI
{
                                /* System initialization handler processing*/
}
```

Figure 5.7 System Initialization Handler Written in C Language

Table 5.6 lists the conditions for system initialization handler processing.

Table 5.6 Conditions for System Initialization Handler Processing

Item	Description
Interrupt mask	Initiated in interrupt mask state (kernel interrupt mask level) Do not change the interrupt mask during system initialization handler execution.
Usable registers	The registers guaranteed in the C language programs (functions) can be used.
Stack pointer	Must be restored when control is returned to the kernel.
Usable system calls	Except for ret_int, system calls that can be issued from the task-independent portion can be used.
Usable stack area	OS stack area is used. Add the stack size to be used by the system initialization handler to the OS stack area. The system initialization stack size must be calculated by using the table to calculate the interrupt handler stack area.
Returning method	Processing must be terminated by the RTS instruction. Restores the stack value at termination.

5.8.2 Defining the System Initialization Handler

The kernel executes the system initialization handler by using the value specified in label _HIPRG_SYSINI as the start address of the system initialization handler.

To define the system initialization handler, specify label _HIPRG_SYSINI as the start address of the system initialization routine. The label must be declared with the export directive.

To cancel the definition of a system initialization routine, label _HIPRG_SYSINI must be defined as 0 by an equate directive and must be declared with the export directive.

The provided system initialization handlers are not defined; label _HIPRG_SYSINI is defined as 0 by an equate directive and is declared with the export directive.

The sample system initialization handler is sample***nnnnz***smp***nnnnz***use.src.

5.9 System Termination Routines

5.9.1 Creating System Termination Routines

The system termination routine is a program that is initiated when a fatal (or critical) error is generated during system execution. The provided system termination routines enter an infinite loop while interrupts are masked at the kernel interrupt mask level.

When the system termination routine is initiated, error information is pushed onto the stack. Refer to the error information in the stack to create a program for each error. There are two stack states for the system termination routine.

Figure 5.8 shows the error information that is pushed onto the stack at an system termination.

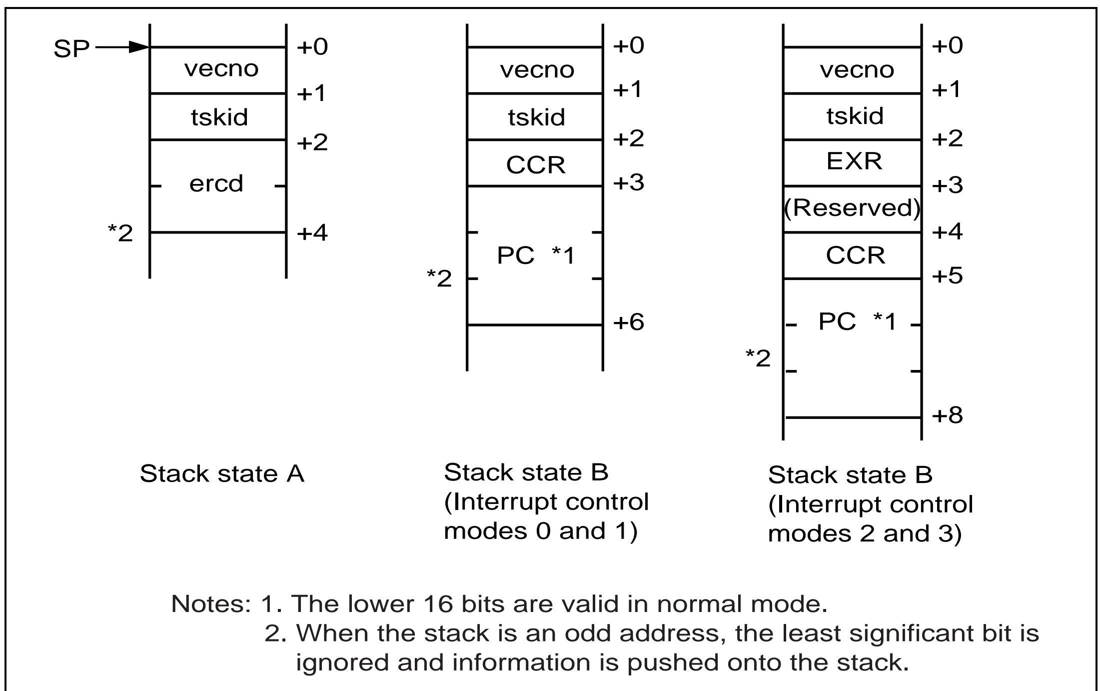


Figure 5.8 Stack State of the System Termination Routine

Table 5.7 shows the causes of system termination and the error information that is pushed onto the stack.

Table 5.7 System Termination Causes

Cause of Termination	vecno	tskid	ercd/Control Register
Setup information error	H'00	H'00	H'0000 to H'0FFF
Timer function not supported	H'00	H'00	H'F9ED
ext_tsk system call issued from task-independent portion	H'00	H'00	H'FFEB
ret_int system call issued while tasks were being executed or while the CPU was being locked	H'00	tskid (H'00 to H'FF)	H'FFBB
Undefined interrupt occurrence	Interrupt vector number	tskid *[1]	CCR, EXR *[2], and PC at error occurrence

Notes: 1. If an error occurs in a task, a task ID is specified in tskid; if an error occurs in a task-independent portion, 0 is specified in tskid.

2. The EXR register is not stored in the stack in interrupt control mode 0 or 1.

Table 5.8 lists invalid setup information and the corresponding error codes (ercd).

Table 5.8 Invalid Setup Information

Error Type	Description	ercd
Invalid address (0 or an odd address)	Kernel stack pointer (_HI_OS_SP) is 0 or an odd address	H'0101
	Timer interrupt stack pointer (_HI_TIM_SP) is 0 or an odd address	H'0102
	Start address of kernel work area (section name: hi8_2s_ram) is 0 or an odd address	H'0103
	TIMCB area (_HI_TIMCB) is 0 or an odd address	H'0104
	TIMCB2 area (_HI_TIMCB2) is 0 or an odd address	H'0105
	TCB area (_HI_TCB) is 0 or an odd address	H'0106
	TCB2 area (_HI_TCB2) is 0 or an odd address	H'0107
	FLGCB area (_HI_FLGCB) is 0 or an odd address	H'0108
	SEMCB area (_HI_SEMCB) is 0 or an odd address	H'0109
	MBXCB area (_HI_MBXCB) is 0 or an odd address	H'010A
	MPFCB area (_HI_MPFCB) is 0 or an odd address	H'010B
	MPLCB area (_HI_MPLCB) is 0 or an odd address	H'010C
	Trace stack pointer (_HI_TRC_SP) is 0 or an odd address	H'010D
	Start address of trace management area (TBACB) is 0 or an odd address	H'010E
	Start address of TIMCB3 area (_HI_TIMCB3) is 0 or an odd address	H'010F
	CYHCB area (_HI_CYHCB) is 0 or an odd address	H'0110
Invalid routine address	Start address of system initialization handler (_HIPRG_SYSINI) is odd address	H'0201
	Start address of timer initialization routine (_HIPRG_TIMINI) is odd address	H'0202
Invalid data setting (out of range)	CPU interrupt control mode (CPUINTM) is 4 or greater	H'0301
	Kernel interrupt mask level (IMASK) is 8 or higher	H'0302
	Maximum priority (MAXPRI) is 32 or higher	H'0303
	Number of defined tasks (TSKCNT) is 256 or more	H'0304
	Number of defined event flags (FLGCNT) is 256 or more	H'0305
	Number of defined semaphores (SEMCNT) is 256 or more	H'0306
	Number of defined mailboxes (MBXCNT) is 256 or more	H'0307
	Number of defined fixed-size memory pools (MPFCNT) is 256 or more	H'0308
	Number of defined variable-size memory pools (MPLCNT) is 256 or more	H'0309
	Number of defined cyclic handlers (CYHCNT) is 256 or more	H'030A

Table 5.8 Invalid Setup Information (cont)

Error Type	Description	ercd
Invalid setup table address (0 or an odd address)	Task definition table (_HI_TDT) is 0 or an odd address	H'0401
	Start address of fixed-size memory pool definition table (_HI_MPFDT) is 0 or an odd address	H'0402
	Start address of variable-size memory pool definition table (_HI_MPLDT) is 0 or an odd address	H'0403
	Start address of undefined interrupt handler (_HI_ILT) is 0 or an odd address	H'0404
	Start address of trace buffer information table (INITRC) is 0 or an odd address	H'0405
	Start address of cyclic handler definition table (_HI_CYCDT) is 0 or an odd address	H'0406
Invalid setup table contents	Initial task priority (ITSKPRI) is 0 or exceeds maximum priority	H'0501
	Task start address (TSKADR) is 0 or an odd address	H'0502
	Task stack pointer (ITSKSP) is 0 or an odd address	H'0503
	Memory block size (BLKLEN) is 0, odd address, or H'FFFA (D'65530) bytes or more	H'0504
	Fixed-size memory pool address (MPF?_TOP) is 0 or an odd address	H'0505
	Variable-size memory pool size is 0, an odd address, or 16 bytes or less	H'0506
	Variable-size memory pool address (MPL?_TOP) is 0 or an odd address	H'0507
	Trace buffer address (TRACE BUFFER ADDRESS) is 0 or an odd address	H'0508
	Start address of cyclic handler is 0 or an odd address	H'0509
	Cyclic timer interval of cyclic handler is 0, or H'80000000 or more	H'050A
	Active information of cyclic handler is illegal (other than 0 or 1)	H'050B

Table 5.8 Invalid Setup Information (cont)

Error Type	Description	ercd
Invalid setup table contents (Extended information)	The number of tasks defined (TSKCNT) and the number of task extended information defined (TSKECNT) do not match	H'0601
	The number of event flags defined (FLGCNT) and the number of event flag extended information defined (FLGECNT) do not match	H'0602
	The number of semaphores defined (SEMCNT) and the number of semaphore extended information defined (SEMECNT) do not match	H'0603
	The number of mailboxes defined (MBXCNT) and the number of mailbox extended information defined (MBXECNT) do not match	H'0604
	The number of fixed-size memory pools defined (MPFCNT) and the number of fixed-size memory pool extended information defined (MPFECNT) do not match	H'0605
	The number of variable-size memory pools defined (MPLCNT) and the number of variable-size memory pool extended information defined (MPLECNT) do not match	H'0606
	The number of cyclic handlers defined (CYHCNT) and the number of cyclic handler extended information defined (CYHECNT) do not match	H'0607

A system termination routine can be written in C language. The processing depends on the user system. When creating a program to restore stack pointers and registers, refer to the descriptions of other handlers.

5.9.2 Defining the System Termination Routine

The system termination routine must always be defined. To define this routine, specify the label _HIPRG_ABNOML as the start address. The label must be declared with the export directive. System termination routine must always be defined.

Note: If label name _HIPRG_ABNOML is specified as 0 or an odd address by an equate directive, normal system operation cannot not be guaranteed.

The sample system termination routine is sample***nnnnz***smp***nnnnz***use.src.

5.10 System Idling Routine

5.10.1 Creating System Idling Routines

When no task is in the READY state, the system makes a transition to the idling state and remains there until a task is activated up by an interrupt.

In the system idling state, the interrupt mask level is specified as open (0). The user can select either a BRA or SLEEP instruction appropriate for the user system to achieve a system idling routine. If the user wishes to use the CPU low-power consumption mode in the system idling state, the SLEEP instruction should be selected.

5.10.2 Defining a System Idling Routine

To define a system idling routine, add label _H_SYSTEM_IDLE at the head of the system idling routine. The label must be declared with the export directive.

A system idling routine must always be defined. If no system idling routine is defined, normal system operation cannot be guaranteed.

The sample system idling routine is sample***nnnnz***smp***nnnnz***use.src.

Section 6 Creating the Setup Table

6.1 Overview

A setup table is created to define the information necessary for system configuration.

The setup table consists of a user definition field and a kernel system definition field. The user definition field is a table that sets the number of defined tasks and the user environment according to the user system to be configured. Modify the user definition field according to the user system environment. The kernel system definition field defines the externally defined symbols and work area used by the kernel.

The system definition field is automatically updated (set) by the user definition field. Therefore, do not modify the system definition field. Otherwise normal system operation is not guaranteed.

The sample setup table file is sample***nnnnz***smp***nnnnz***sup.src.

6.2 User Definition Field

The fields to be defined in the user definition field are described below using the H8S/2655 advanced mode as an example.

Constant Definition Field: The constant definition field defines information required for synchronization and communication and for time management.

Task Definition Field: The task definition field defines information required for task execution.

Fixed-Size Memory Pool Definition Field: The fixed-size memory pool definition field defines information required for fixed-size memory pools.

Variable-Size Memory Pool Definition Field: The variable-size memory pool definition field defines information required for variable-size memory pools.

Cyclic Handler Definition Field: The cyclic handler definition field defines information required for cyclic handlers.

System Call Trace Function Definition Field: The system call trace function definition field defines information required for system call trace functions.

Extended Information Definition Field: The extended information definition field defines information required for extended information for tasks, event flags, semaphores, mailboxes, fixed-size and variable-size memory pools, and cyclic handlers.

Note: The above items must be set regardless of each field definition. If not, an undefined error will occur in the linking stage.

6.2.1 Defining the Constant Definition Field

This field defines information required for the kernel functions (such as synchronization-and-communication and time-management functions). The following items are defined in the definition field in the sample setup table.

Table 6.1 summarizes the information to be defined in the constant definition field.

Table 6.1 Information Defined in Constant Definition Field

Information	Label Name	Definition Information	Notes
Interrupt control mode	CPUINTM*	Mode = 3	
Kernel interrupt mask level	IMASK*	Level = 6	
Maximum task priority	MAXPRI*	Lowest priority = 31	
Number of event flags defined	FLGCNT*	Maximum event flag ID = 4	
Number of semaphores defined	SEMCNT*	Maximum semaphore ID = 4	
Number of mailboxes defined	MBXCNT*	Maximum mailbox ID = 4	
OS stack size	OSSTKSIZ*	OS stack size = 52	18 + 20 + 6 + 8
Timer stack size	TIMSTKSIZ*	Timer driver stack size = 64	40 + 10 + 6 + 8
Trace stack size	TRCSTKSIZ*	Trace function stack size = 40	26 + 6 + 8
Timeout function defined	TTMOUT*	USE	

Note: These label names must not be modified because the kernel refers to them. If they are modified, normal system operation cannot be guaranteed.

Figure 6.1 shows the calculation of stack size. For details, refer to appendix A, Memory Size. The timer stack and trace stack size can be calculated in the same way.

Table A.2 OS Stack Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Stack area used by OS	18 (Advanced mode)	18	Always necessary
Stack area for interrupts	10 x LOWiNTNST [*1] + 6 x UPPiNTNST [*2]	10 x 2 + 6 x 1	When interrupt control mode 3 is used, Number of interrupt nests equal to or lower than IMASK = 2, Number of interrupt nests equal to or higher than IMASK = 1
Stack area for undefined interrupt [*3]	8	8	An undefined interrupt has occurred
Total		52	18 + 20 + 6 + 8

Notes: 1. Number of interrupt nests that are equal to or lower than the kernel interrupt mask level.
2. Number of interrupt nests that are higher than the kernel interrupt mask level (including NMIs).
3. Required when an undefined interrupt occurs.

Figure 6.1 OS Stack Area Calculation

Figure 6.2 shows the constant definition field. Only modify the bold-italic face. Otherwise normal system operation cannot be guaranteed.

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    VALUE define section                                             %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;LABEL                  VALUE    ;:[ RANGE ]      ;: COMMENT
;---------------------------------------------------------------------------
CPUINTM:        .assign    3                                                 (1)


IMASK:          .assign    6                                                 (2)


MAXPRI:         .assign    31                                                (3)

FLGCNT:         .assign    4                                                 (4)

SEMCNT:         .assign    4                                                 (5)

MBXCNT:         .assign    4                                                 (6)


OSSTKSIZ:       .equ       18+(10*2)+(6*1)+8                                 (7)

TIMSTKSIZ:      .equ       40+(10*1)+(6*1)+8                                 (8)


TRCSTKSIZ:      .equ       26+(6*1)+8                                        (9)
;

TTMOUT:         .assign    USE                                               (10)
;
```

Figure 6.2 Constant Definition Field

Notes

(1) Defines CPU interrupt control modes 0 to 3. For details on the interrupt control mode, refer to the target MCU hardware manual

(2) Defines the kernel interrupt mask level. Defined values differ from those in the CPU interrupt control mode (CPUINTM).

CPUINTM	IMASK Value
0	0 or 1
1	0 to 3
2	0 to 7
3	0 to 8

An interrupt whose level is higher than the kernel interrupt mask level is always accepted without delay.

(3) Defines the lowest task priority of the system to be created. MAXPRI ranges from 1 to 31, with a higher value indicating lower priority.

(4) Defines the maximum event flag ID. FLGCNT ranges from 0 to 255. If FLGCNT is 0, no event flags will be defined.

(5) Defines the maximum semaphore ID. SEMCNT ranges from 0 to 255. If SEMCNT is 0, no semaphores will be defined.

(6) Defines the maximum mailbox ID. MBXCNT ranges from 0 to 255. If MBXCNT is 0, no mailboxes will be defined.

(7) Defines the OS stack area size. For details, refer to appendix A, Memory Size.

(8) Defines the timer driver stack area size. For details, refer to appendix A, Memory Size. If the timer driver is not used, the timer stack size must be set to 0.

(9) Defines the trace function stack area size. For details, refer to appendix A, Memory Size. If the trace function is not used, the timer stack size must be set to 0.

(10) Defines whether the timeout function is used.
USE: Used
NOTUSE: Not used
If TIMSTKSIZ is 0, the timeout function becomes invalid.

6.2.2 Defining Task

This field defines information to register tasks. The sample setup table registers:

- Five tasks (IDs 1 to 5)

  Tasks 1 and 2 are tutorial tasks for the debugging extension (DX) and are defined. Tasks 3 to 5 are not defined.
- Task stack

  Minimum stack size: 86

  Tasks 4 and 5 use the same stack as a shared stack.

Table 6.2 shows the contents of the task definitions.

Table 6.2 Contents of Task Definition

Task ID	Start Address	Initial State	Initial Priority	Stack Pointer	Stack Size Used by Task
1	TASKA	DORMANT	1	TSK1_SP	36 bytes
2	TASKB	DORMANT	2	TSK2_SP	36 bytes
3	None (0)	NON-EXISTENT	3	TSK3_SP	32 bytes
4	None (0)	NON-EXISTENT	4	TSK4_SP (Shared)	32 bytes
5	None (0)	NON-EXISTENT	5	TSK4_SP (Shared)	32 bytes

Figure 6.3 shows task definition field. Only modify the bold-italic face. Otherwise normal system operation cannot be guaranteed.

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    TASK define section                                              %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;               TASK_TOP_LABEL                  ;: COMMENT
;---------------------------------------------------------------------------
        .import _TASKA                                                          (1)
        .import _TASKB
;
;------ Usage ---------------------------------------------------------------
;               .res.b  SIZE +TSKSTKSIZ
;TSK?_SP_LABEL: .equ        $
;---------------------------------------------------------------------------
TSKSTKSIZ:      .equ        50+(10*2)+(6*1)+6+8                                 (2)
```

Figure 6.3 Task Definition Field

```
        .section        h2sstack,stack,align=2
                .res.b  (36) +TSKSTKSIZ                                          (3)
TSK1_SP:        .equ    $
                .res.b  8

                .res.b  (36) +TSKSTKSIZ
TSK2_SP:        .equ    $
                .res.b  8

                .res.b  (32) +TSKSTKSIZ
TSK3_SP:        .equ    $
                .res.b  8

                .res.b  (32) +TSKSTKSIZ
TSK4_SP:        .equ    $
                .res.b  8
;
        .section        h2ssetup,code,align=2
_HI_H8S:        .res.b  10                      ;: System Area
;------ Usage -------------------------------------------------------------
;LABEL          .data.b IMOD, ITSKPRI           ;: COMMENT
;               .data.l ITSKADR, ITSKSP         ;: COMMENT
;--------------------------------------------------------------------------
NOEXS:          .assign 0
RDY:            .assign 1
DMT:            .assign (-1)
TDTLEN:         .assign    10;<- Not Change !
        .section        h2ssetup,code,align=2
_HI_TDT:        .equ    $-TDTLEN
TDT_TOP:        .equ    $
tdt_id1:        .data.b DMT, 1                                                   (4)
                .data.l _TASKA, TSK1_SP

tdt_id2:        .data.b DMT, 2
                .data.l _TASKB, TSK2_SP

tdt_id3:        .data.b NOEXS, 3
                .data.l 0, TSK3_SP

tdt_id4:        .data.b NOEXS, 4
                .data.l 0, TSK4_SP

tdt_id5:        .data.b NOEXS, 5
                .data.l 0, TSK4_SP
TDT_BTM:
TSKCNT:         .equ    (TDT_BTM-TDT_TOP)/TDTLEN
```

Figure 6.3 Task Definition Field (cont)

Notes

(1) Declares the start address of the task to be used as an external reference symbol.

(2) Defines the minimum stack size used by a stack. The minimum stack size does not include the stack size used by each task. For details, refer to appendix A, Memory Size.

(3) [Task stack size must be defined for each stack pointer]

Defines the task stack size.

Line 1: Defines the stack size used

Stack size = Stack size used by each task + minimum stack size

Line 2: Defines the label

(Task stack bottom)

Line 3: Defines the shared-stack-management area

When using the shared-stack function, define 8 bytes for use by the management area in the direction of ascending addresses. If the shared stack function is not used, this area need not be defined.

(4) [Task information must be defined for each task]

Defines task information.

[Format]

```
LABEL   .data.b    IMOD, ITSKPRI
        .data.l    ITSKADR, ITSKSP
```

— LABEL: Can be freely defined (Can be omitted)

— IMOD (Definition/initiation requests)

Specifies each task's initial state at task definition and system initiation as follows:

- NOEXS (=0): Undefined
- RDY (=1): READY state
- DMT (other than 0 or 1): DORMANT state

— ITSKPRI (Initial priority)

Defines each task's initial priority. ITSKPRI ranges from 1 to MAXPRI (priority number definition).

— ITSKADR (Task start address)

Defines the start address of a task.

— ITSKSP (Task stack pointer)

Defines the end address of a stack area used by the task.

6.2.3 Defining Fixed-Size Memory Pools

This field defines information to register fixed-size memory pools. The sample setup table registers:

- Four fixed-size memory pools (IDs 1 to 4)

Table 6.3 shows the contents of fixed-size memory pool definitions.

Table 6.3 Contents of Fixed-Size Memory Pool Definitions

Memory Pool ID	Number of Memory Blocks	Memory Block Size	Label Name
1	14	12 bytes	MPF1_TOP
2	14	12 bytes	MPF2_TOP
3	14	12 bytes	MPF3_TOP
4	14	12 bytes	MPF4_TOP

Figure 6.4 shows a fixed-size memory pool definition field. Only modify the bold-italic font. Otherwise normal system operation cannot be guaranteed.

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    FIXED-SIZE MEMORYPOOL define section                              %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage --------------------------------------------------------------
;MB?_CNT_LABEL: .assign VALUE   ;:[  RANGE  ]   ;: COMMENT
;MB?_LEN_LABEL: .assign VALUE   ;:[  RANGE  ]   ;: COMMENT
;--------------------------------------------------------------------------
MB1_CNT:        .assign 14                                                     (1)
MB1_LEN:        .assign 12
;
MB2_CNT:        .assign 14
MB2_LEN:        .assign 12
;
MB3_CNT:        .assign 14
MB3_LEN:        .assign 12
;
MB4_CNT:        .assign 14
MB4_LEN:        .assign 12
;
;------ Usage --------------------------------------------------------------
;MPF?_TOP_LABEL:.res.b  MEMORYPOOL_SIZE         ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2smpf,data,align=2
MPF1_TOP:       .res.b  MB1_CNT * (MB1_LEN + 4)                                (2)

MPF2_TOP:       .res.b  MB2_CNT * (MB2_LEN + 4)

MPF3_TOP:       .res.b  MB3_CNT * (MB3_LEN + 4)

MPF4_TOP:       .res.b  MB4_CNT * (MB4_LEN + 4)
;
```

Figure 6.4 Fixed-Size Memory Pool Definition Field

```
;------ Usage ------------------------------------------------------------
;LABEL           .data.w BLFCNT, BLFLEN          ;: COMMENT
;               .data.l MPF_TOP_ADDRESS         ;: COMMENT
;------------------------------------------------------------------------
MPFDTLEN:       .assign 8;<- Not Change !       ;: MPFDT Length
        .section        h2ssetup,code,align=2
_HI_MPFDT:      .equ    $-MPFDTLEN              ;: Fixed-size MemoryPool define
table
MPFDT_TOP:      .equ    $
mpfdt_id1:      .data.w MB1_CNT, MB1_LEN                                    (3)
                .data.l MPF1_TOP

mpfdt_id2:      .data.w MB2_CNT, MB2_LEN
                .data.l MPF2_TOP

mpfdt_id3:      .data.w MB3_CNT, MB3_LEN
                .data.l MPF3_TOP

mpfdt_id4:      .data.w MB4_CNT, MB4_LEN
                .data.l MPF4_TOP
MPFDT_BTM:
MPFCNT:         .equ    (MPFDT_BTM-MPFDT_TOP)/MPFDTLEN
```

Figure 6.4 Fixed-Size Memory Pool Definition Field (cont)

Notes

(1) [The number of memory blocks and the memory block size must be specified for each memory block]

MB?_CNT and MB?_LEN specify the number of memory blocks and the memory block size.

Labels such as MB1_CNT and MB1_LEN are used to define the memory pool area to specify the memory pool definition table.

(2) [A memory pool area must be defined for each memory pool]

Defines the fixed-size memory pool area. A label must be specified as the start address for each memory pool area. The label name is MPF?_TOP in the sample setup table.

In the example, each memory pool size can be defined using the following expression.

Fixed-size memory pool size = MB?_CNT x (MB?_LEN + 4)

If fixed-size memory pool is unnecessary, delete all the lines shown in a bold-italic font in (2) in this figure.

(3) [Memory pool information must be defined for each memory pool]

Defines fixed-size memory pool information.

[Format]

```
LABEL    .data.w    BLFCNT, BLFLEN
         .data.l    MPF_TOP_ADDRESS
```

— LABEL: Can be freely defined. (Can be omitted.)
— BLFCNT: Number of memory blocks
— BLFLEN: Fixed-size memory block size
— MPF_TOP_ADDRESS: Start address of the fixed-size memory pool

Specify 0 for BLFCNT when not using the fixed-size memory information.

If fixed-size memory pool is unnecessary, delete all the lines shown in a bold-italic font in (3) in this figure.

6.2.4 Defining Variable-Size Memory Pools

This field defines information to register variable-size memory pools. The sample setup table defines the following variable-size memory pools:

- Four variable-size memory pools (IDs 1 to 4)

Table 6.4 shows the contents of variable-size memory pool definition.

Table 6.4 Contents of Variable-Size Memory Pool Definitions

Task ID	Memory Block Size	Label Name
1	380 bytes	MPL1_TOP
2	380 bytes	MPL2_TOP
3	380 bytes	MPL3_TOP
4	380 bytes	MPL4_TOP

Figure 6.5 shows a variable-size memory pool definition field. Only modify the bold-italic face. Otherwise normal system operation cannot be guaranteed.

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    VARIABLE-SIZE MEMORYPOOL define section                            %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ------------------------------------------------------------
;MPL?_SIZ_LABEL:.assign   VALUE    ;:[  RANGE  ]   ;: COMMENT
;-------------------------------------------------------------------------
MPL1_SIZ:       .assign 380                                                    (1)

MPL2_SIZ:       .assign 380

MPL3_SIZ:       .assign 380

MPL4_SIZ:       .assign 380
;
;------ Usage ------------------------------------------------------------
;MPL?_TOP_LABEL:.res.b VARIABLE_MEMORYPOOL_SIZE ;: COMMENT
;-------------------------------------------------------------------------
        .section        h2smpl,data,align=2
MPL1_TOP:      .res.b  MPL1_SIZ                                                (2)

MPL2_TOP:      .res.b  MPL2_SIZ

MPL3_TOP:      .res.b  MPL3_SIZ

MPL4_TOP:      .res.b  MPL4_SIZ
;
;------ Usage ------------------------------------------------------------
;LABEL          .data.l BLKSIZ                  ;: COMMENT
;               .data.l VARIABLE_MEMORYPOOL_TOP ;: COMMENT
;-------------------------------------------------------------------------
MPLDTLEN:      .assign 8;<- Not Change !        ;: MPLDT Length
        .section        h2ssetup,code,align=2
_HI_MPLDT:      .equ    $-MPLDTLEN
MPLDT_TOP:      .equ    $
mpldt_id1:      .data.l MPL1_SIZ                                               (3)
                .data.l MPL1_TOP

mpldt_id2:      .data.l MPL2_SIZ
                .data.l MPL2_TOP

mpldt_id3:      .data.l MPL3_SIZ
                .data.l MPL3_TOP

mpldt_id4:      .data.l MPL4_SIZ
                .data.l MPL4_TOP
MPLDT_BTM:
MPLCNT:         .equ    (MPLDT_BTM-MPLDT_TOP)/MPLDTLEN
```

Figure 6.5 Variable-size Memory Pool Definition Field

Notes

(1) [The memory block size must be defined for each memory block]

Defines the variable-size memory block size. The label is used to define the memory pool area to specify the memory pool definition table.

When specifying the variable-size memory pool, specify a size including the 16-byte kernel management area. Specification size = Memory pool size to be used + (16 x maximum number of blocks acquired)

(2) [A memory pool area must be defined for each memory pool]

Defines the variable-size memory pool area. A label must be specified as the start address for each memory pool area. The label name is MPL?_TOP in the sample setup table.

If variable-size memory pool is unnecessary, delete all the lines shown in a bold-italic font in (2) in this figure.

(3) [Variable-size memory information must be defined for each memory pool]

Defines variable-size memory pool information.

[Format]

```
LABEL   .data.l    BLKSIZ
        .data.l    MPL_TOP_ADDRESS
```

— LABEL: Can be freely defined. (Can be omitted.)

— BLKSIZ: The memory block size

— MPL_TOP_ADDRESS: Start address of the variable-size memory pool

Specify 0 for BLKSIZ and MPL_TOP_ADDRESS when not using the variable-size memory information.

If variable-size memory pool is unnecessary, delete all the lines shown in a bold-italic font in (3) in this figure.

6.2.5 Defining Cyclic Handlers

This field defines information to register cyclic handlers. In the sample setup table, it is assumed no cyclic handlers are defined.

- When the debugging extension (DX) is not used (definition of the sample files)
  — Number of cyclic handlers: 4 (cyclic handler specification numbers 1 to 4 are not defined)
- When the debugging extension (DX) is used (definition of the sample files)
  Cyclic handler specification number 5 is defined when the debugging extension is used
  — Number of cyclic handlers: 5 (cyclic handler specification numbers 1 to 4 are not defined)
  The Debug Daemon will be defined as cyclic handler specification number 5

Table 6.5 shows the contents of cyclic handler definitions of the sample files.

Table 6.5 Contents of Cyclic Handler Definitions

Cyclic Handler Specification Number	Activation State	Invoked Interval	Label Name
1	OFF	0	None (NADR)
2	OFF	0	None (NADR)
3	OFF	0	None (NADR)
4	OFF	0	None (NADR)
5	ON	5	HI_DEAMON_MAIN

- Example of cyclic handler definition (Example of cyclic handler definition in figure 6.6)

  The definition contents of the sample cyclic handler as follows

  — The symbol (_CYCHDR) of the cyclic handler address is declared (imported) as the external reference symbol.

  — The cyclic handler information is defined

  - Cyclic handler specification number: 6
  - Cyclic handler activation state: CYHON (activated)
  - Cyclic handler timer interval: 10
  - Cyclic handler address: _CYCHDR

  If a cyclic handler is added, the extended information must be added.

Figure 6.6 shows an example of a definition of a cyclic handler definition field. In this figure, a cyclic handler (cyclic handler specification number 6) has been added to the sample file. Only modify the bold-italic face. Otherwise normal system operation cannot be guaranteed.

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    cyclic handler define section                                      %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;       .import CYCHDR_TOP_LABEL                ;: COMMENT
;----------------------------------------------------------------------------
        .import _CYCHDR                                                       (1)
;
;------ Usage ---------------------------------------------------------------
;LABEL:         .data.w CYC_ACTIVATE            ;: COMMENT
;               .data.l CYC_TIME, CYCHDR_TOP    ;: COMMENT
;----------------------------------------------------------------------------
CYHOFF          .assign 0                       ;:initial cycact data = OFF
CYHON           .assign 1                       ;:initial cycact data = ON
CYHDTLEN        .assign 10;<-Dont't Change!     ;:CYHDT length
;
_HI_CYHDT:      .equ    $-CYHDTLEN
CYHDT_TOP:      .equ    $
cyhdt_no1:      .data.w CYHOFF                                                (2)
                .data.l 0, NADR

cyhdt_no2:      .data.w CYHOFF
                .data.l 0, NADR

cyhdt_no3:      .data.w CYHOFF
                .data.l 0, NADR

cyhdt_no4:      .data.w CYHOFF
                .data.l 0, NADR

                .aifdef DX
cyhdt_no5:      .data.w CYHON
                .data.l 5, HI_DEAMON_MAIN
                .aendi
cyhdt_no6:      .data.w CYHON                                                 (3)
                .data.l 10, _CYCHDR
CYHDT_BTM:
CYHCNT:         .equ    (CYHDT_BTM-CYHDT_TOP)/CYHDTLEN
```

Figure 6.6 Definition Example of Cyclic Handler Definition Field

Notes

(1) Declares (imports) the start address of the cyclic handler as the external reference symbol. This is an example of cyclic handler definition.

(2) [Cyclic handler information must be defined for each cyclic handler]

Defines cyclic handler information.

[Format]

```
LABEL    .data.w    CYC_ACTIVATE
         .data.l    CYC_TIME, CYCHDR_TOP
```

— LABEL: Can be freely defined. (Can be omitted.)

— CYC_ACTIVATE (Specifies the cyclic handler activation state.)

Defines the cyclic handler activation state at system initiation.

- CYHOFF (=0): Not initiated (not activated)
- CYHON (=1): Initiated (activated)

— CYC_TIME (Cyclic time interval)

Specifies the cycle time to initiate the cyclic handler.

— CYCHDR_TOP (Cyclic handler address)

Specifies the start address of the handler to define. If NADR(–1) is specified, the cyclic handler ID will not be defined.

If cyclic handler is unnecessary, delete all the lines shown in a bold-italic font in (2) in this figure.

If the debugging extension is used, the debug daemon cyclic handler will be defined.

(3) For cyclic handler specification number 6, cyclic handler activation state is specified as activated (CYHON), cyclic time interval is specified as 10, and cyclic handler address is specified as symbol (_CYCHDR). This is an example of cyclic handler definition.

6.2.6 Defining Trace Functions

This field defines information to register trace functions. The sample setup table registers:

- A maximum of four trace information acquisitions

Figure 6.7 shows a trace function definition field. Only modify the bold-italic face. Otherwise normal system operation cannot be guaranteed.

```
;------ Usage ------------------------------------------------------------
;TRC_CNT:.assign TRACE COUNT
;TRC_BUF:.assign TRACE BUFFER ADDRESS
;------------------------------------------------------------------------
        .section        h2strc,data,align=2
TRC_CNT:        .assign 4                                                 (1)
TRC_BUF:        .res.b  16+(TRC_CNT*28)                                   (2)
;
;------ Usage -------------------------------------
;INITRC         .data.l TRACE BUFFER ADDRESS
;               .data.w TRACE COUNT
;-------------------------------------------------
        .section        h2ssetup,code,align=2
INITRC:         .equ    $
                .data.l TRC_BUF                                           (3)
                .data.w TRC_CNT
```

Figure 6.7 Trace Function Definition Field

Notes

(1) Defines the maximum amount of trace information that can be acquired by the trace function. Specify 0 if the trace function is not used.
(2) Defines the trace buffer area. In the sample example, the trace buffer area size is calculated as follows:

    Trace buffer area size = 16 + TRC_CNT x 28

    If the trace function is not used, write this line as a comment.
(3) Start address of the trace buffer area. Specify 0 if the trace function is not used.

6.2.7 Defining Extended Information

Extended information can be defined for the following objects: tasks, event flags, semaphores, mailboxes, fixed-size memory pools, variable-size memory pools, and cyclic handlers.

The extended information can be defined freely by the user for each ID of the resource concerning the target object.

The extended information is a packet of memory area reserved to enter information concerning the target object. The start address of the packet is specified as the extended information.

In the sample program, H'0 is specified as the start address for the extended information. In the start address (H'0) of the sample extended information, extended information is not specified.

Figure 5.8 shows an extended information definition field. Only modify the bold-italic font. Otherwise normal system operation cannot be guaranteed.

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Task Extended Information define section                        %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;LABEL          .data.l TSK?_EXINF              ;: COMMENT
;---------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_TSKEXINF:   .equ    $-EXLEN
TSKE_TOP:       .equ    $
tsk1_exinf:     .data.l 00000000                                        (1)

tsk2_exinf:     .data.l 00000000

tsk3_exinf:     .data.l 00000000

tsk4_exinf:     .data.l 00000000

tsk5_exinf:     .data.l 00000000

TSKE_BTM:
TSKECNT:        .equ    (TSKE_BTM-TSKE_TOP)/EXLEN
                                ;:[0...255]     ;: tsk exinf count
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Event Flag Extended Information define section                  %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage ---------------------------------------------------------------
;LABEL          .data.l FLG?_EXINF              ;: COMMENT
;---------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_FLGEXINF:   .equ    $-EXLEN
FLGE_TOP:       .equ    $
flg1_exinf:     .data.l 00000000                                        (2)

flg2_exinf:     .data.l 00000000

flg3_exinf:     .data.l 00000000

flg4_exinf:     .data.l 00000000


FLGE_BTM:
FLGECNT:        .equ    (FLGE_BTM-FLGE_TOP)/EXLEN
```

Figure 6.8 Extended Information Definition Field

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Semaphore Extended Information define section                  %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;LABEL          .data.l SEM?_EXINF              ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_SEMEXINF:   .equ    $-EXLEN
SEME_TOP:       .equ    $
sem1_exinf:     .data.l 00000000                                            (3)

sem2_exinf:     .data.l 00000000

sem3_exinf:     .data.l 00000000

sem4_exinf:     .data.l 00000000

SEME_BTM:
SEMECNT:        .equ    (SEME_BTM-SEME_TOP)/EXLEN
                                ;:[0...255]     ;: sem exinf count
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Mailbox Extended Information define section                    %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------
;LABEL          .data.l MBX?_EXINF              ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MBXEXINF:   .equ    $-EXLEN
MBXE_TOP:       .equ    $
mbx1_exinf:     .data.l 00000000                                            (4)

mbx2_exinf:     .data.l 00000000

mbx3_exinf:     .data.l 00000000

mbx4_exinf:     .data.l 00000000


MBXE_BTM:
MBXECNT:        .equ    (MBXE_BTM-MBXE_TOP)/EXLEN
```

Figure 6.8 Extended Information Definition Field (cont)

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Fixed-size MemoryPool Extended Information define section      %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage --------------------------------------------------------------
;LABEL           .data.l MPF?_EXINF              ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MPFEXINF:   .equ    $-EXLEN
MPFE_TOP:       .equ    $
mpf1_exinf:     .data.l 00000000                                        (5)

mpf2_exinf:     .data.l 00000000

mpf3_exinf:     .data.l 00000000

mpf4_exinf:     .data.l 00000000


MPFE_BTM:
MPFECNT:        .equ    (MPFE_BTM-MPFE_TOP)/EXLEN
                                ;:[0...255]     ;: mpf exinf count
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Variable-size MemoryPool Extended Information define section   %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage --------------------------------------------------------------
;LABEL           .data.l MPL?_EXINF              ;: COMMENT
;--------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_MPLEXINF:   .equ    $-EXLEN
MPLE_TOP:       .equ    $
mpl1_exinf:     .data.l 00000000                                        (6)

mpl2_exinf:     .data.l 00000000

mpl3_exinf:     .data.l 00000000

mpl4_exinf:     .data.l 00000000


MPLE_BTM:
MPLECNT:        .equ    (MPLE_BTM-MPLE_TOP)/EXLEN
;
```

Figure 6.8 Extended Information Definition Field (cont)

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;%%%    Cyclic Handler Extended Information define section            %%%
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;------ Usage -------------------------------------------------------------------
;LABEL          .data.l CYH?_EXINF              ;: COMMENT
;-------------------------------------------------------------------------------
        .section        h2ssetup,code,align=2
_HI_CYCEXINF:   .equ    $-EXLEN
CYHE_TOP:       .equ    $
cyh1_exinf:     .data.l 00000000                                              (7)

cyh2_exinf:     .data.l 00000000

cyh3_exinf:     .data.l 00000000

cyh4_exinf:     .data.l 00000000

                .aifdef DX
cyh5_exinf:     .data.l 00000000
                .aendi
CYHE_BTM:
CYHECNT:        .equ    (CYHE_BTM-CYHE_TOP)/EXLE
```

Figure 6.8 Extended Information Definition Field (cont)

Notes

(1) Defines task extended information.

[Format]

`LABEL    .data.l    TSK?_EXINF`

— LABEL: Can be freely defined. (Can be omitted.)

— TSK?_EXINF: (Extended information)

An address can be defined.

Note: Task extended information must be defined for the number of tasks as defined in section 6.2.2, Defining Task.
If the number of task extended information values does not match the number of tasks defined, the system will terminate abnormally. If extended information is unnecessary, delete all the lines shown in a bold-italic font in (1) in this figure.

(2) Defines event flag extended information.

[Format]

`LABEL    .data.l    FLG?_EXINF`

— LABEL: Can be freely defined. (Can be omitted.)

— FLG?_EXINF: (Extended information)

An address can be defined.

Note: Event flag extended information must be defined for the maximum event flag definition number (FLGCNT) as defined in section 6.2.1, Defining the Constant Definition Field. If the number of event flag extended information values does not match the maximum number of event flags defined, the system will terminate abnormally. If extended information is unnecessary, delete all the lines shown in a bold-italic font in (2) in this figure.

(3) Defines semaphore extended information.

[Format]

```
LABEL   .data.l    SEM?_EXINF
```

— LABEL: Can be freely defined. (Can be omitted.)

— SEM?_EXINF: (Extended information)

An address can be defined.

Note: Semaphore extended information must be defined for the maximum semaphore definition number (SEMCNT) as defined in section 6.2.1, Defining the Constant Definition Field.
If the number of semaphore extended information values does not match the maximum number of semaphores defined, the system will terminate abnormally. If extended information is unnecessary, delete all the lines shown in a bold-italic font in (3) in this figure.

(4) Defines mailbox extended information.

[Format]

```
LABEL   .data.l    MBX?_EXINF
```

— LABEL: Can be freely defined. (Can be omitted.)

— MBX?_EXINF: (Extended information)

An address can be defined.

Note: Mailbox extended information must be defined for the maximum mailbox definition number (MBXCNT) as defined in section 6.2.1, Defining the Constant Definition Field. If the number of mailbox extended information values does not match the maximum number of mailboxes defined, the system will terminate abnormally. If extended information is unnecessary, delete all the lines shown in a bold-italic font in (4) in this figure.

(5) Defines fixed-size memory pool extended information.

[Format]

```
LABEL   .data.l    MPF?_EXINF
```

— LABEL: Can be freely defined. (Can be omitted.)

— MPF?_EXINF: (Extended information)

An address can be defined.

Note: Fixed-size memory pool extended information must be defined for the fixed-size definition number as defined in section 6.2.3, Defining Fixed-size Memory Pools.
If the number of fixed-size memory pool extended information values does not match the number of fixed-size memory pools defined, the system will terminate abnormally.
If extended information is unnecessary, delete all the lines shown in a bold-italic font in (5) in this figure.

(6) Defines variable-size memory pool extended information.

[Format]

```
LABEL   .data.l    MPL?_EXINF
```

— LABEL: Can be freely defined. (Can be omitted.)

— MPL?_EXINF: (Extended information)
An address can be defined.

Note: Variable-size memory pool extended information must be defined for the variable-size definition number as defined in section 6.2.4, Defining Variable-size Memory Pools.
If the number of variable-size memory pool extended information values does not match the number of variable-size memory pools defined, the system will terminate abnormally. If extended information is unnecessary, delete all the lines shown in a bold-italic font in (6) in this figure.

(7) Defines cyclic handler extended information.

[Format]

```
LABEL   .data.l    CYH?_EXINF
```

— LABEL: Can be freely defined. (Can be omitted.)

— CYH?_EXINF: (Extended information)
An address can be defined.

Note: Cyclic handler extended information must be defined for the cyclic handler definition number as defined in section 6.2.5, Defining Cyclic Handlers.
If the number of cyclic handler extended information values does not match the number of cyclic handlers defined, the system will terminate abnormally. If extended information is unnecessary, delete all the lines shown in a bold-italic font in (7) in this figure. If the debugging extension is used, the debug daemon cyclic handler will be defined.

6.3 System Definition Field

This field defines externally defined symbols used by the kernel, constants, and the kernel system work area.

Figure 6.9 shows the system definition field. In the system definition field, symbols can be automatically defined from the value defined in the user definition field.

Do not modify the system definition field. Otherwise correct system operation cannot be guaranteed.

```
        .include        "setup.inc"                    Includes system
                                                       definition file.
;
;*******************************************;
        .end; of 2655asUP.MAR
```

Figure 6.9 System Definition Field

For details on the system work area reserved by the system definition field, refer to appendix A, Memory Size.

Section 7 Creating the Interrupt Vector Table

7.1 Overview

The interrupt vector table defines the start address of each interrupt handler, so that control passes to the appropriate interrupt processing when an interrupt occurs. If no interrupt handler start address is defined for a vector number, an undefined interrupt handler start address must be defined for that vector number.

The sample interrupt vector table file is sample***nnnnz***smp***nnnnz***ili.src.

Create an interrupt vector for the user system by referring to the provided files.

7.2 Defining Interrupt Handler

An interrupt handler can be used by defining the interrupt handler start address to the corresponding vector number in the interrupt vector table.

The following interrupt handlers are defined in the sample vector table (table 6.3).

- CPU: H8S/2655
- CPU operating mode: advanced mode

Table 7.1 lists the defined interrupt handlers.

Table 7.1 Defined Interrupt Handlers

No	Interrupt Handler	Label Name	Vector Number	Notes
1	CPU initialization routine	_H_2S_CPUINI	0	Power-on reset
2	CPU initialization routine	_H_2S_CPUINI	1	Manual reset
3	Timer interrupt handler	_H_2S_TIM	32	TPUch0
4	Undefined interrupt handler	_H_2SINT??	??	??: Vector number of other than items 1 to 3.

For details on the causes of interrupts, refer to the refer to the target MCU hardware manual.

Figure 7.1 shows a coding example from the H8S/2655 series interrupt vector table 2655avec.src for the advanced mode provided as a sample file.

```
;*********************************************************************
;***                                                              ***
;***    HI2000/3 Version (uITRON V3.0)                            ***
;***    HI2000/3 vector table                                     ***
;***                                                              ***
;***    Copyright (c)  Hitachi, Ltd. 1998.                        ***
;***    Licensed Material of Hitachi, Ltd.                        ***
;***                                                              ***
;*********************************************************************
        .program        _2655avec
        .heading        "### 2655avec.src : for H8S/2655 ###"
        .section        h2svectr,code,locate=0
;
        .import         _H_2SINT00,_H_2SINT01,_H_2SINT02,_H_2SINT03,_H_2SINT04(1)
        .import         _H_2SINT05,_H_2SINT06,_H_2SINT07,_H_2SINT08,_H_2SINT09
        .import         _H_2SINT10,_H_2SINT11,_H_2SINT12,_H_2SINT13,_H_2SINT14
        .import         _H_2SINT15,_H_2SINT16,_H_2SINT17,_H_2SINT18,_H_2SINT19
        .import         _H_2SINT20,_H_2SINT21,_H_2SINT22,_H_2SINT23,_H_2SINT24
        .import         _H_2SINT25,_H_2SINT26,_H_2SINT27,_H_2SINT28,_H_2SINT29
        .import         _H_2SINT30,_H_2SINT31,_H_2SINT32,_H_2SINT33,_H_2SINT34
        .import         _H_2SINT35,_H_2SINT36,_H_2SINT37,_H_2SINT38,_H_2SINT39
        .import         _H_2SINT40,_H_2SINT41,_H_2SINT42,_H_2SINT43,_H_2SINT44
        .import         _H_2SINT45,_H_2SINT46,_H_2SINT47,_H_2SINT48,_H_2SINT49
        .import         _H_2SINT50,_H_2SINT51,_H_2SINT52,_H_2SINT53,_H_2SINT54
        .import         _H_2SINT55,_H_2SINT56,_H_2SINT57,_H_2SINT58,_H_2SINT59
        .import         _H_2SINT60,_H_2SINT61,_H_2SINT62,_H_2SINT63,_H_2SINT64
        .import         _H_2SINT65,_H_2SINT66,_H_2SINT67,_H_2SINT68,_H_2SINT69
        .import         _H_2SINT70,_H_2SINT71,_H_2SINT72,_H_2SINT73,_H_2SINT74
        .import         _H_2SINT75,_H_2SINT76,_H_2SINT77,_H_2SINT78,_H_2SINT79
        .import         _H_2SINT80,_H_2SINT81,_H_2SINT82,_H_2SINT83,_H_2SINT84
        .import         _H_2SINT85,_H_2SINT86,_H_2SINT87,_H_2SINT88,_H_2SINT89
        .import         _H_2SINT90,_H_2SINT91
;
        .import         _H_2S_CPUINI    ;: in 'cpuini'                          (2)
        .import         _H_2S_TIM       ;: in 'h2suser'
;
```

Figure 7.1 Coding Example from the Interrupt Vector Table 2655avec.src

```
;**************************************************************************
;*specifications ;                                                       *
;*name      = h2svectr : H8S/2655 vector table for HI2000/3         ;    *
;*function  = 1. h8 interrupt handler address define for hi8        ;    *
;*          = 2. h8 exception handler address define for hi8        ;    *
;*          = 3. hi8 system standard support module                 ;    *
;*          =    (1) reset     : "_H_2S_CPUINI"  for power on       ;    *
;*          = 4. hi8 system standard support module                 ;    *
;*          =    (1) TPU0 tgi0a: "_H_2S_TIM"   for system timer     ;    *
;*date      = 99/02/22                                              ;    *
;*author    = Hitachi, Ltd.                                         ;    *
;*attribute = public                                                ;    *
;*class     = unit                                                  ;    *
;*linkage   = h8 vector table top address = h'0000 for HI2000/3     ;    *
;*input     = none                                                  ;    *
;*output    = none                                                  ;    *
;*parameter = er7 : stack pointer                                   ;    *
;***********  CPU interrupt mode = 3  ******************************;    *
;*                            |            |                        ;    *
;*    er7(stack pointer) --> +0 +------------+                      ;    *
;*                            |    EXR     |                        ;    *
;*                         +1 +------------+                        ;    *
;*                            |  reserved  |                        ;    *
;*                         +2 +------------+                        ;    *
;*                            |    CCR     |                        ;    *
;*                         +3 +------------+                        ;    *
;*                            |            |                        ;    *
;*                            +            +                        ;    *
;*                            |     PC     |                        ;    *
;*                            +            +                        ;    *
;*                            |            |                        ;    *
;*                         +6 +------------+                        ;    *
;*end of specifications ;                                                 *
;**************************************************************************
;
        .radix  d                       ;:xxxxx -> d'xxxxx
;
;-------.data.l < address >             ;: H8S/2655 vector no. contents
        .data.l _H_2S_CPUINI ;_H_2SINT00  ;: vector no.00 <reset>          (3)
        .data.l _H_2S_CPUINI ;_H_2SINT01  ;: vector no.01 <reset>
        .data.l _H_2SINT02              ;: vector no.02 [reserve]
        .data.l _H_2SINT03              ;: vector no.03 [reserve]
        .data.l _H_2SINT04              ;: vector no.04 [reserve]
        .data.l _H_2SINT05              ;: vector no.05 [reserve]
```

Figure 7.1 Coding Example from the Interrupt Vector Table 2655avec.src (cont)

```
        .data.l _H_2SINT06              ;: vector no.06 [reserve]
        .data.l _H_2SINT07              ;: vector no.07 <NMI        >
        .data.l _H_2SINT08              ;: vector no.08 <TRAPA  #0  >
        .data.l _H_2SINT09              ;: vector no.09 <TRAPA  #1  >
        .data.l _H_2SINT10              ;: vector no.10 <TRAPA  #2  >
        .data.l _H_2SINT11              ;: vector no.11 <TRAPA  #3  >
        .data.l _H_2SINT12              ;: vector no.12 [reserve]
        .data.l _H_2SINT13              ;: vector no.13 [reserve]
        .data.l _H_2SINT14              ;: vector no.14 [reserve]
        .data.l _H_2SINT15              ;: vector no.15 [reserve]
        .data.l _H_2SINT16              ;: vector no.16 <IRQ0       >
        .data.l _H_2SINT17              ;: vector no.17 <IRQ1       >
        .data.l _H_2SINT18              ;: vector no.18 <IRQ2       >
        .data.l _H_2SINT19              ;: vector no.19 <IRQ3       >
        .data.l _H_2SINT20              ;: vector no.20 <IRQ4       >
        .data.l _H_2SINT21              ;: vector no.21 <IRQ5       >
        .data.l _H_2SINT22              ;: vector no.22 <IRQ6       >
        .data.l _H_2SINT23              ;: vector no.23 <IRQ7       >
        .data.l _H_2SINT24              ;: vector no.24 <SWDTEND    >
        .data.l _H_2SINT25              ;: vector no.25 <WOVI       >
        .data.l _H_2SINT26              ;: vector no.26 <CMI        >
        .data.l _H_2SINT27              ;: vector no.27 [reserve]
        .data.l _H_2SINT28              ;: vector no.28 <ADI        >
        .data.l _H_2SINT29              ;: vector no.29 [reserve]
        .data.l _H_2SINT30              ;: vector no.30 [reserve]
        .data.l _H_2SINT31              ;: vector no.31 [reserve]
        .data.l _H_2S_TIM  ;_H_2SINT32  ;: vector no.32 <TGI0A  tpu0 >          (4)
        .data.l _H_2SINT33              ;: vector no.33 <TGI0B  tpu0 >
        .data.l _H_2SINT34              ;: vector no.34 <TGI0C  tpu0 >
        .data.l _H_2SINT35              ;: vector no.35 <TGI0D  tpu0 >
        .data.l _H_2SINT36              ;: vector no.36 <TCI0V  tpu0 >
        .data.l _H_2SINT37              ;: vector no.37 [reserve]
        .data.l _H_2SINT38              ;: vector no.38 [reserve]
        .data.l _H_2SINT39              ;: vector no.39 [reserve]
        .data.l _H_2SINT40              ;: vector no.40 <TGI1A  tpu1 >
        .data.l _H_2SINT41              ;: vector no.41 <TGI1B  tpu1 >
        .data.l _H_2SINT42              ;: vector no.42 <TCI1V  tpu1 >
        .data.l _H_2SINT43              ;: vector no.43 <TCI1U  tpu1 >
        .data.l _H_2SINT44              ;: vector no.44 <TGI2A  tpu2 >
        .data.l _H_2SINT45              ;: vector no.45 <TGI2B  tpu2 >
        .data.l _H_2SINT46              ;: vector no.46 <TCI2V  tpu2 >
        .data.l _H_2SINT47              ;: vector no.47 <TCI2U  tpu2 >
        .data.l _H_2SINT48              ;: vector no.48 <TGI3A  tpu3 >
        .data.l _H_2SINT49              ;: vector no.49 <TGI3B  tpu3 >
        .data.l _H_2SINT50              ;: vector no.50 <TGI3C  tpu3 >
        .data.l _H_2SINT51              ;: vector no.51 <TGI3D  tpu3 >
        .data.l _H_2SINT52              ;: vector no.52 <TCI3V  tpu3 >
```

Figure 7.1 Coding Example from the Interrupt Vector Table 2655avec.src (cont)

```
        .data.l _H_2SINT53              ;: vector no.53 [reserve]
        .data.l _H_2SINT54              ;: vector no.54 [reserve]
        .data.l _H_2SINT55              ;: vector no.55 [reserve]
        .data.l _H_2SINT56              ;: vector no.56 <TGI4A  tpu4 >
        .data.l _H_2SINT57              ;: vector no.57 <TGI4B  tpu4 >
        .data.l _H_2SINT58              ;: vector no.58 <TCI4V  tpu4 >
        .data.l _H_2SINT59              ;: vector no.59 <TCI4U  tpu4 >
        .data.l _H_2SINT60              ;: vector no.60 <TGI5A  tpu5 >
        .data.l _H_2SINT61              ;: vector no.61 <TGI5B  tpu5 >
        .data.l _H_2SINT62              ;: vector no.62 <TCI5V  tpu5 >
        .data.l _H_2SINT63              ;: vector no.63 <TCI5U  tpu5 >
        .data.l _H_2SINT64              ;: vector no.64 <CMIA0       >
        .data.l _H_2SINT65              ;: vector no.65 <CMIB0       >
        .data.l _H_2SINT66              ;: vector no.66 <OVI0        >
        .data.l _H_2SINT67              ;: vector no.67 [reserve]
        .data.l _H_2SINT68              ;: vector no.68 <CMIA1       >
        .data.l _H_2SINT69              ;: vector no.69 <CMIB1       >
        .data.l _H_2SINT70              ;: vector no.70 <OVI1        >
        .data.l _H_2SINT71              ;: vector no.71 [reserve]
        .data.l _H_2SINT72              ;: vector no.72 <DEND0A dmac >
        .data.l _H_2SINT73              ;: vector no.73 <DEND0B dmac >
        .data.l _H_2SINT74              ;: vector no.74 <DEND1A dmac >
        .data.l _H_2SINT75              ;: vector no.75 <DEND1B dmac >
        .data.l _H_2SINT76              ;: vector no.76 [reserve]
        .data.l _H_2SINT77              ;: vector no.77 [reserve]
        .data.l _H_2SINT78              ;: vector no.78 [reserve]
        .data.l _H_2SINT79              ;: vector no.79 [reserve]
        .data.l _H_2SINT80              ;: vector no.80 <ERI0   sci0 >
        .data.l _H_2SINT81              ;: vector no.81 <RXI0   sci0 >
        .data.l _H_2SINT82              ;: vector no.82 <TXI0   sci0 >
        .data.l _H_2SINT83              ;: vector no.83 <TEI0   sci0 >
        .data.l _H_2SINT84              ;: vector no.84 <ERI1   sci1 >
        .data.l _H_2SINT85              ;: vector no.85 <RXI1   sci1 >
        .data.l _H_2SINT86              ;: vector no.86 <TXI1   sci1 >
        .data.l _H_2SINT87              ;: vector no.87 <TEI1   sci1 >
        .data.l _H_2SINT88              ;: vector no.88 <ERI2   sci2 >
        .data.l _H_2SINT89              ;: vector no.89 <RXI2   sci2 >
        .data.l _H_2SINT90              ;: vector no.90 <TXI2   sci2 >
        .data.l _H_2SINT91              ;: vector no.91 <TEI2   sci2 >
;
;*************************************;
        .end; of 2655avec.src
```

Figure 7.1 Coding Example from the Interrupt Vector Table 2655avec.src (cont)

Notes

(1) Declares the start routine of the undefined interrupt handler as the external reference symbol (No.4 in table 7.1).

(2) Declares the start routine of the interrupt handler to be defined as the external reference symbol (Nos. 1 to 3 in table 7.1).

(3) Defines the CPU initialization routine. This routine must be defined (Nos. 1 and 2 in table 7.1).

(4) Defines the timer interrupt handler. This handler must be defined when system calls wai_flg, set_tim, and get_tim, and txxx_xxx system calls (such as twai_sem) are used (No. 3 in table 7.1).

Section 8 Load Module Creation

8.1 Overview

The Hitachi Embedded Workshop (HEW) is used to create load modules. Refer to the HEW's manual or on-line help system to find out how to use the HEW.

Creating a system involves compiling and linking together the following four types of files into a load module; the kernel library, the setup table, the interrupt vector table, and the application files.

Figure 8.1 shows the flow for creating a load module.

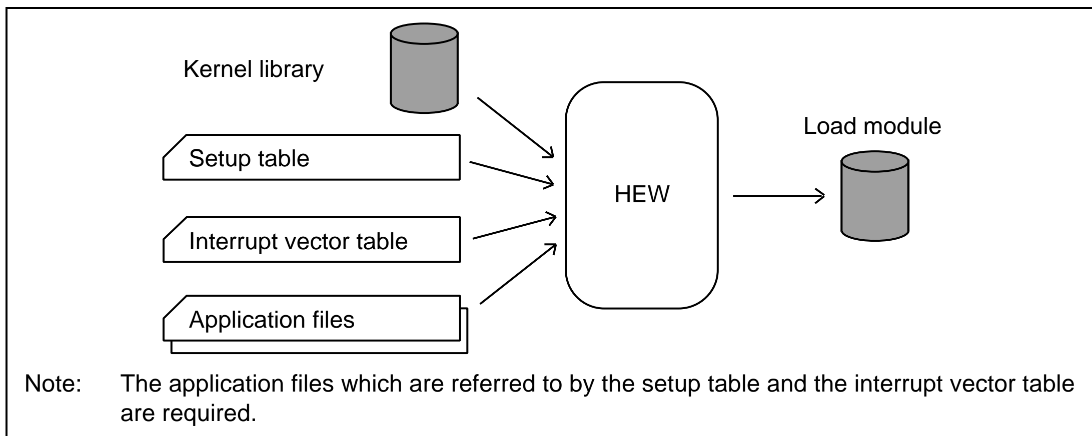


Figure 8.1 Creating a Load Module

Kernels for the H8S/2600 and H8S/2000 CPUs are available. Each kernel has two MCU operating modes: the advanced mode and the normal mode. Select the kernel and the supplied application files according to the environment.

8.2 Workspace and Project Files

Create load modules with the HEW according to the following procedure.

1. Add the files necessary to create the load module to a project.
2. Specify the options for the C compiler, the assembler, and the inter-module optimizer.
3. Run the Build command.

The HI2000/3 provides a sample workspace file “product.hws”. Double-click the “product.hws” filename to activate the HEW with the workspace “product”.

The workspace “product” contains sample projects corresponding to a variety of devices. As shown in table 8.1, there are four sample projects that correspond to two CPUs in two operating modes. Select the project which matches your environment (CPU and operating mode) and refer to the descriptions on the following pages.

Select a project in the HEW’s workspace window then select [Set as Current Project] from the pop-up menu, as shown in figure 8.2. Projects for unused environments can be deleted.

If any CPU other than the H8S/2655 or H8S/2245 is to be used, the project must first be selected, then the files for system construction that have been added to the project must be changed so that they suit the CPU.

Table 8.1 Sample Projects

Project Name	Configuration*	Description
hi26a	hi26a	Load module for the H8S/2600 CPU advanced mode (already set for the H8S/2655)
hi26n	hi26n	Load module for the H8S/2600 CPU normal mode (already set for the H8S/2655)
hi20a	hi20a	Load module for the H8S/2000 CPU advanced mode (already set for the H8S/2245)
hi20n	hi20n	Load module for the H8S/2000 CPU normal mode (already set for the H8S/2245)

Note: The default settings create a load module within the given configuration.

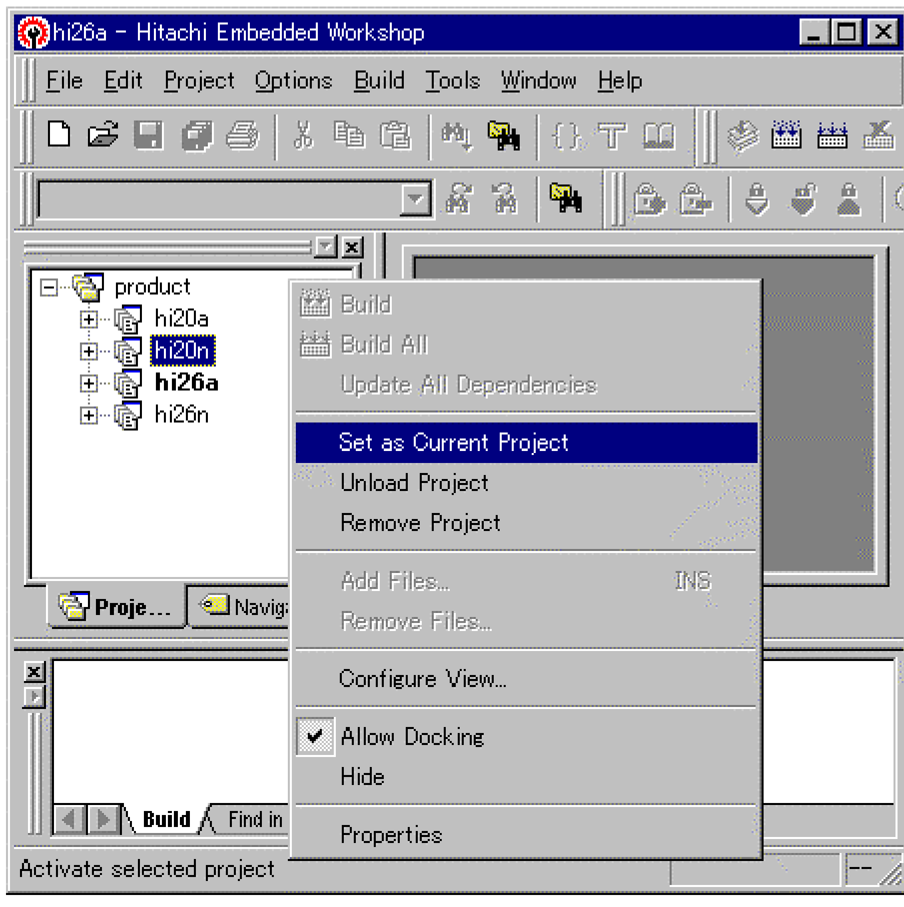


Figure 8.2 Selecting a Project

When the 'Build' command is executed on a selected sample project, the load module is created by executing the compiler, assembler, and inter-module optimizer in sequence.

8.3 Load Module Creation

8.3.1 Adding Files to a Project

Table 8.2 lists the files that are required for a project. The sample project files at shipment are for the H8S/2655 and H8S/2245.

If a CPU other than the H8S/2655 or H8S/2245 is used, add new system configuration files and delete the old ones.

Table 8.2 Files Required for Project

File Name	Description	Notes
Application files	Tasks and interrupt handlers	
sample***nnnnz***smp***nnnnz***sup.src	Setup table	Always necessary
sample***nnnnz***smp***nnnnz***use.src	Timer driver	
	System termination routine	Always necessary
	System idling routine	Always necessary
	System initialization handler	
sample***nnnnz***smp***nnnnz***vec.src	Interrupt vector table	Always necessary
sample***nnnnz***smp***nnnnz***ili.src	Undefined interrupt handler	
sample***nnnnz***smp***nnnnz***cpu.src	CPU initialization routine	Always necessary
sample\task.c	DX tutorial task	

Note: ***nnnn*** (italic-bold face) corresponds to a device.
z (italic-bold face) shows the operating mode (a: advanced mode. n: normal mode).

Add files to the project by using the following procedure.

1. Start the HEW and open the sample workspace.
2. Select the project which corresponds to the environment to be used.
3. Select [Add Files] from the File menu and add the application files.
4. If a CPU other than the H8S/2655 or H8S/2245 is used, add new system configuration files to the project.
5. Refer to the options of the system configuration files that have already been added and set the options for the newly added files. After setting the options, delete the system configuration files that will not be used.

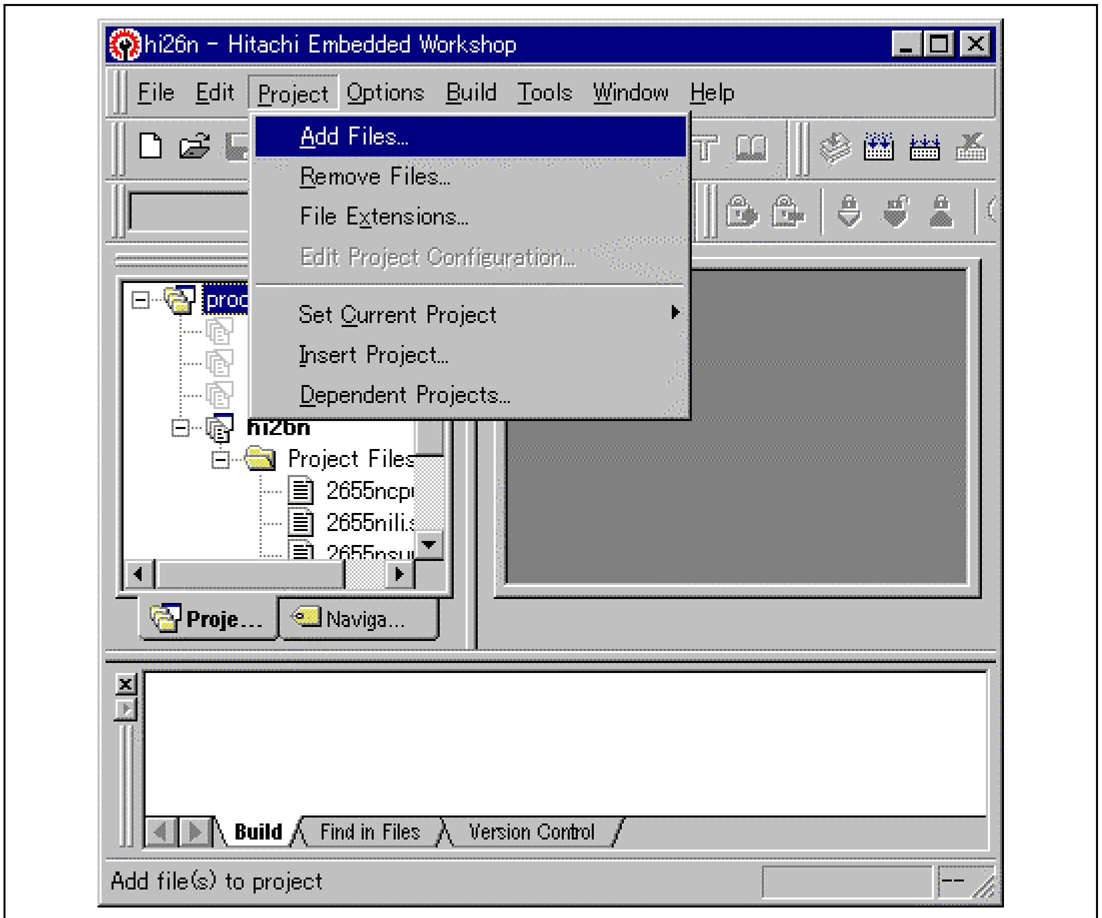


Figure 8.3 Adding Files to the Project

8.3.2 Compiler and Assembler Options

For details on the compiler and assembler options, refer to section B, Compiler and Assembler Options.

Refer to table 8.3 and figures 8.4 to 8.9 to set the compiler and assembler options for the system configuration files.

Table 8.3 Compiler and Assembler Options

File Name[*]	Option
Common to all system configuration files	CPU tab • Specify according to the CPU used Object tabs • Output file directory: $(CONFIGDIR) • Debug Information: Specifies the output of debugging information List tab: • Specifies no list output
sample***nnnnz***smp***nnnnz***sup.src	Source tab • Include file directories: Specify $(PROJDIR)\sample for the directory • Defines: DX=Action (If the Debugging Extension is installed)
sample***nnnnz***smp***nnnnz***use.src	—
sample***nnnnz***smp***nnnnz***vec.src	—
sample***nnnnz***smp***nnnnz***ili.src	—
sample***nnnnz***smp***nnnnz***cpu.src	Source tab • Defines: DX=Action (If the Debugging Extension is installed)
sample\task.c	Source tab • Include file directories: Specify $(PROJDIR)\sample for the directory Object tab • Section: Specify P = Ptask or B = Btask

Note: ***nnnn*** (italic-bold face) corresponds to the device.
z (italic-bold face) shows the operating mode (a: advanced mode. n: normal mode).

An example of the CPU tab common to all system configuration files is shown in figure 8.4.

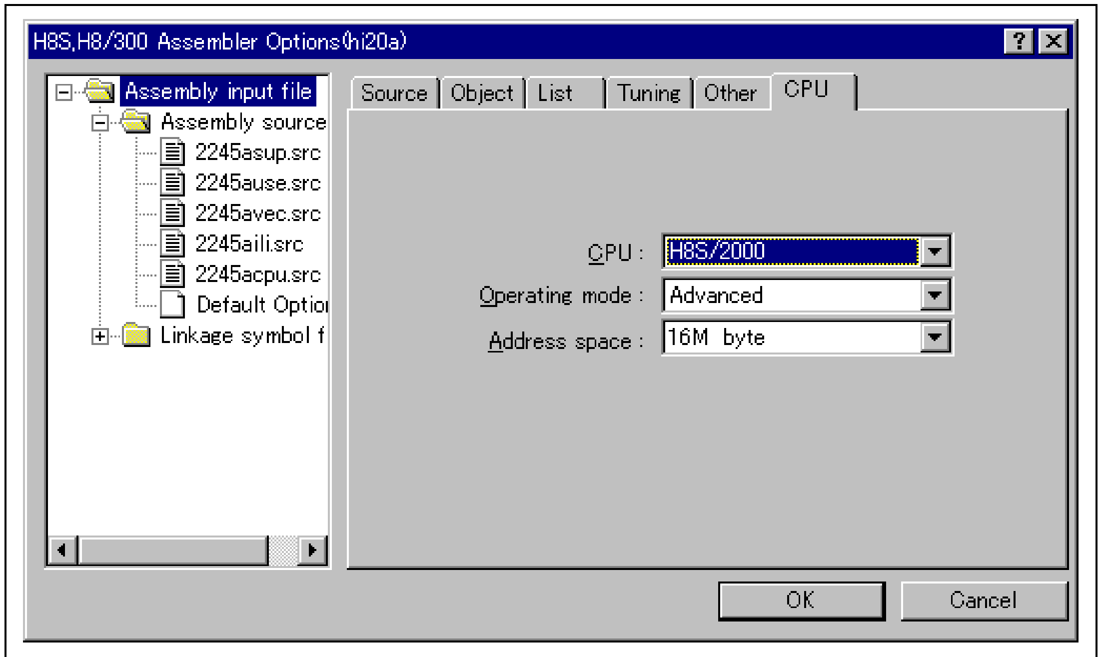


Figure 8.4 CPU Tab Window in the H8S, H8/300 Assembler Options

An example of the Object tab common to all system configuration files is shown in figure 8.5.

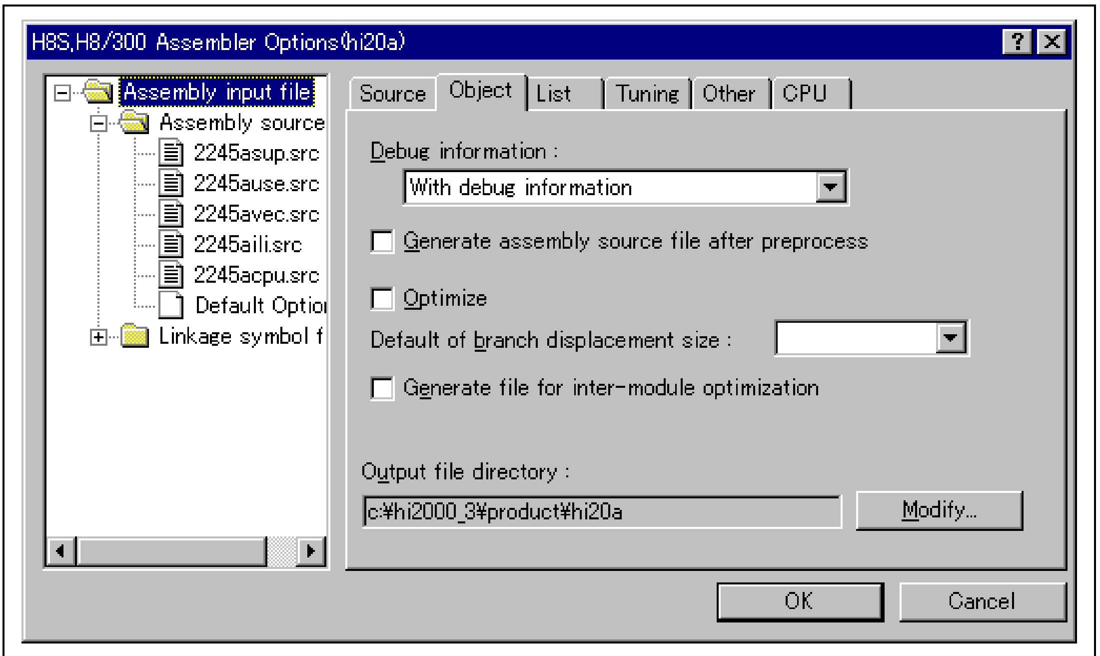


Figure 8.5 Object Tab Window in the H8S, H8/300 Assembler Options

An example of the List tab common to all system configuration files is shown in figure 8.6.

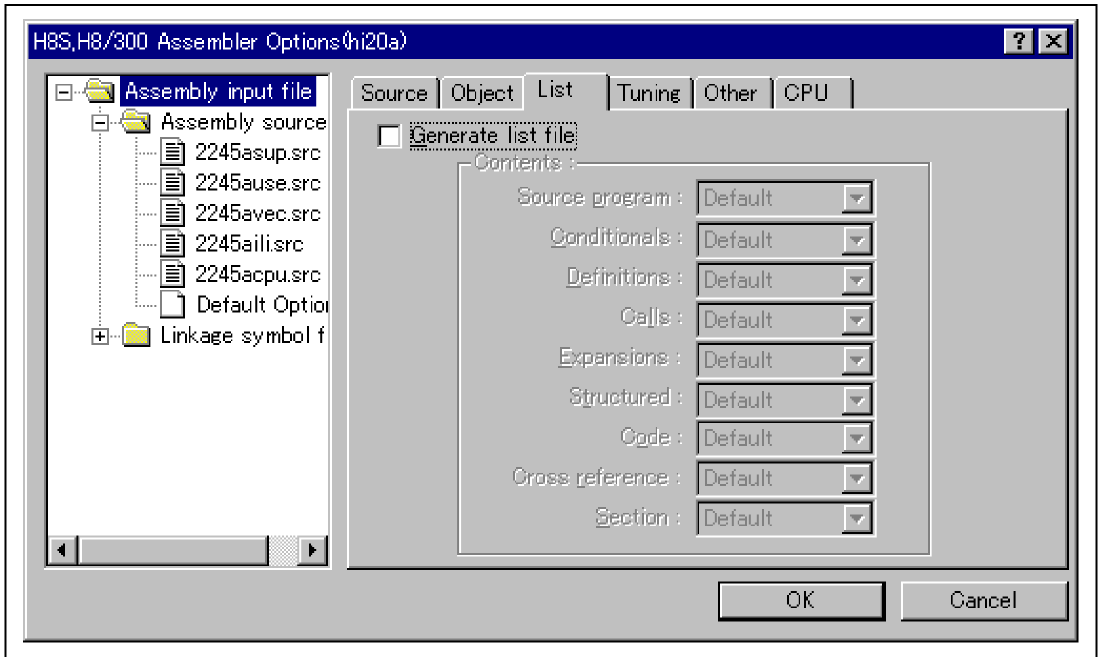


Figure 8.6 List Tab Window in the H8S, H8/300 Assembler Options

Note: “Select the ‘Source’ tab to specify include-file directories or define symbols”.

Figure 8.7 shows an example of the specification of include files on the tabbed page ‘Source’. “Include file directories” is selected from the dropdown list labelled “Show entries for”.

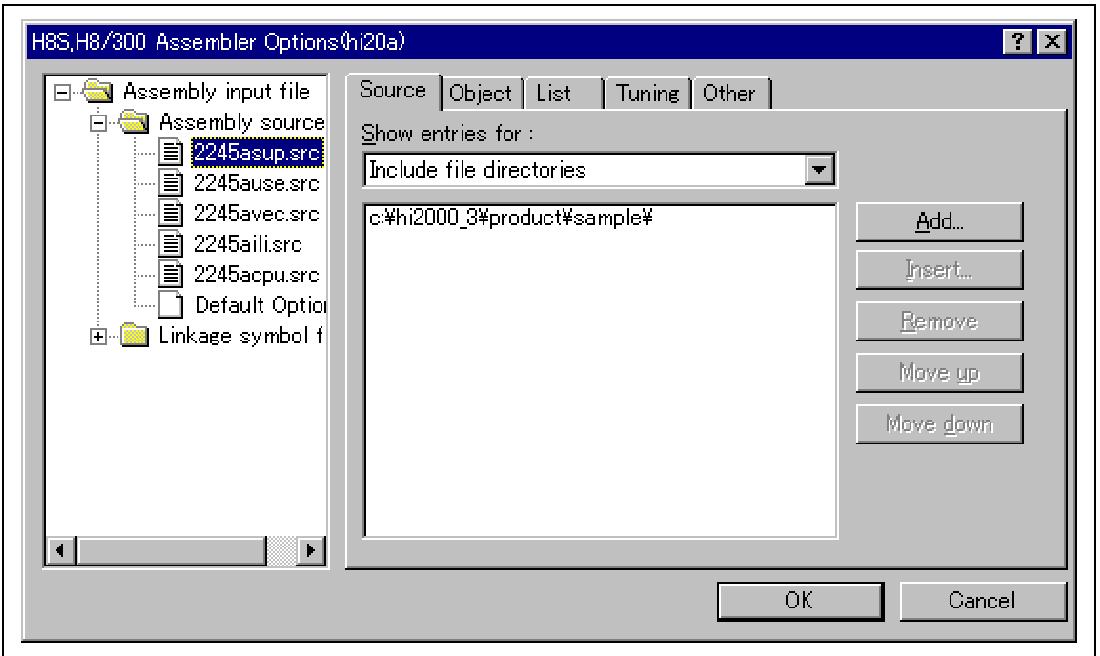


Figure 8.7 Source Tab Window in the H8S, H8/300 Assembler Options (Include file directories)

Figure 8.8 shows an example of the specification of the debugging extension (DX) on the tabbed page ‘Source’. ‘Defines’ is selected from the dropdown list labelled “Show entries for”.

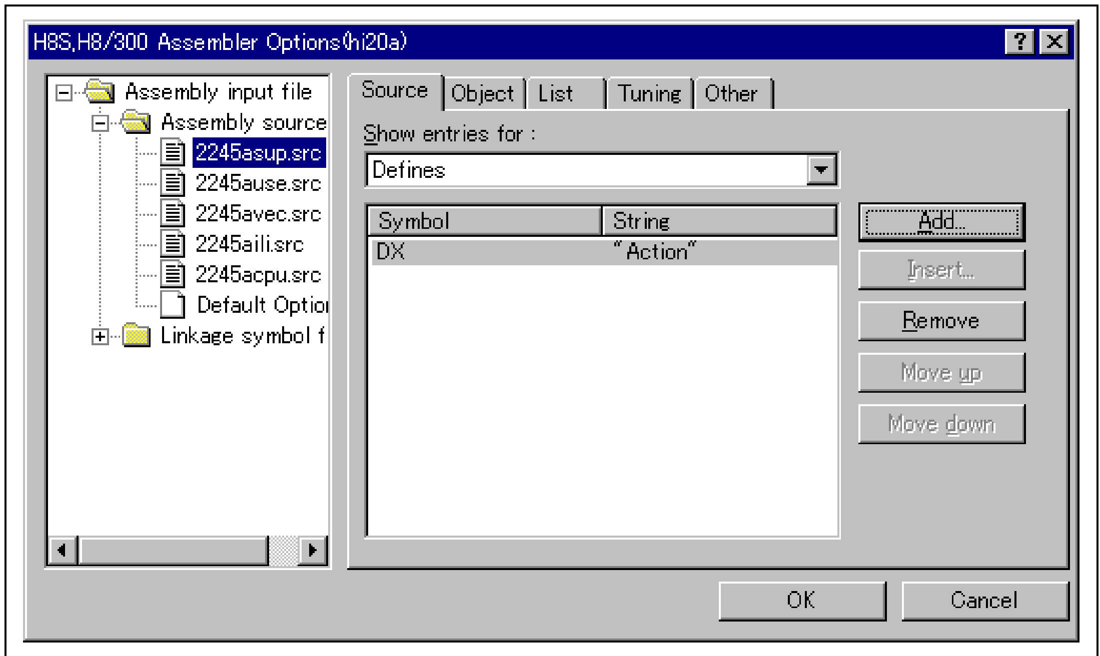


Figure 8.8 Source Tab Window in the H8S, H8/300 Assembler Options (Defines)

Figure 8.9 shows an example of the specification of sections on the tabbed page “Object”. A section is selected from the dropdown list labelled “Section”.

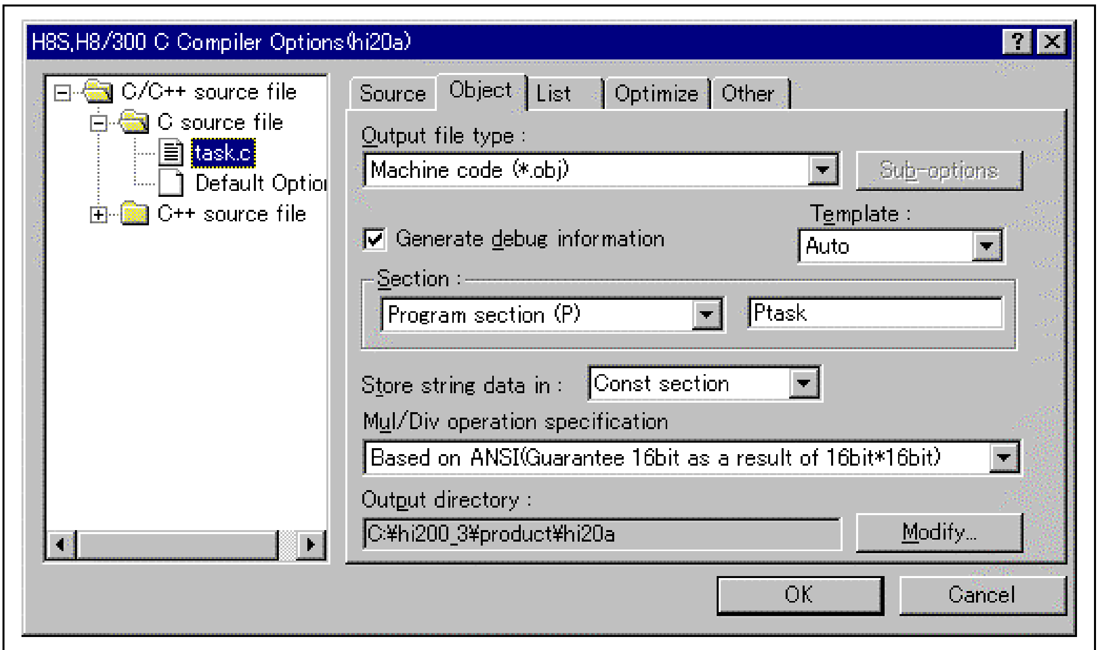


Figure 8.9 Object Tab Window in the H8S, H8/300 C Compiler Options

8.3.3 Inter-Module Optimizer Setting

1. Inter-Module Optimizer Options Input Tab

   In figure 8.10, a kernel library, which has a parameter check function and a shared-stack function, and a C-language interface library is specified for the provided project file. Specify library files according to the user environment (CPU and operating mode).

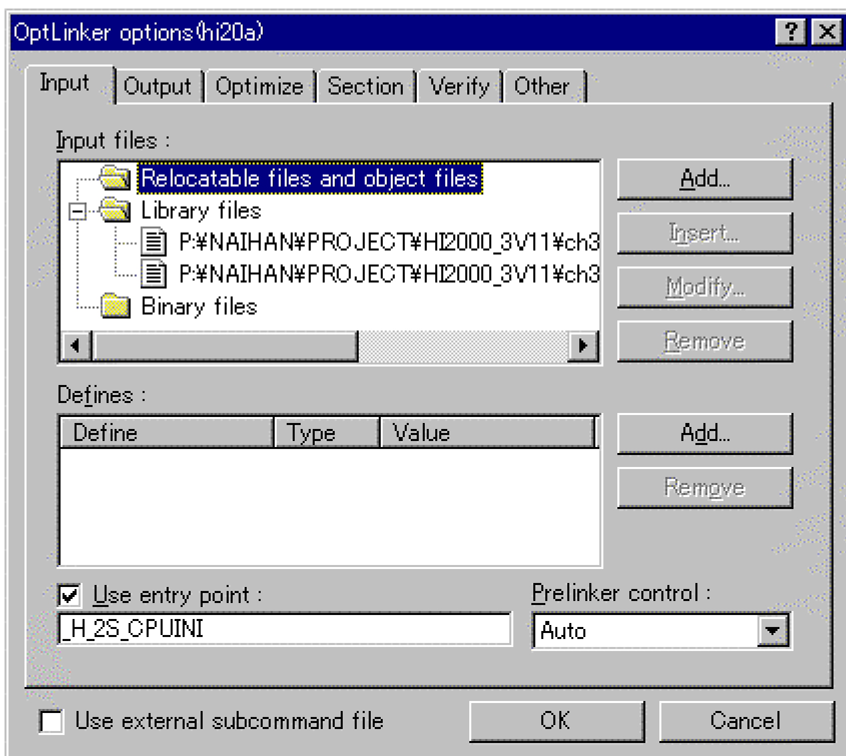


Figure 8.10 Inter-Module Optimizer Options Input Tab

Kernel libraries and C-language interface libraries can be selected from table 8.4.

If application libraries or standard libraries provided by the H8S series C compiler is used, they must be specified through this Input tab.

Table 8.4 Supplied Library File List

Library Names			File Name	Parameter Check Function	Shared-Stack Function
Kernel library	H8S/2600 CPU	Advanced mode	hilib\26aknlps.lib	Yes	Yes
			hilib\26aknlpn.lib	Yes	No
			hilib\26aknlns.lib	No	Yes
			hilib\26aknlnn.lib	No	No
		Normal mode	hilib\26nknlps.lib	Yes	Yes
			hilib\26nknlpn.lib	Yes	No
			hilib\26nknlns.lib	No	Yes
			hilib\26nknlnn.lib	No	No
	H8S/2000 CPU	Advanced mode	hilib\20aknlps.lib	Yes	Yes
			hilib\20aknlpn.lib	Yes	No
			hilib\20aknlns.lib	No	Yes
			hilib\20aknlnn.lib	No	No
		Normal mode	hilib\20nknlps.lib	Yes	Yes
			hilib\20nknlpn.lib	Yes	No
			hilib\20nknlns.lib	No	Yes
			hilib\20nknlnn.lib	No	No
System-call C-language interface library	H8S/2600 CPU	Advanced mode	hilib\26acif.lib	—	—
		Normal mode	hilib\26ncif.lib	—	—
	H8S/2000 CPU	Advanced mode	hilib\20acif.lib	—	—
		Normal mode	hilib\20ncif.lib	—	—

2. Inter-Module Optimizer Options Output Tab

The Output tab specifies the format and type of load module, debugging information, and load module path. The projects provided produce load modules within the configuration.

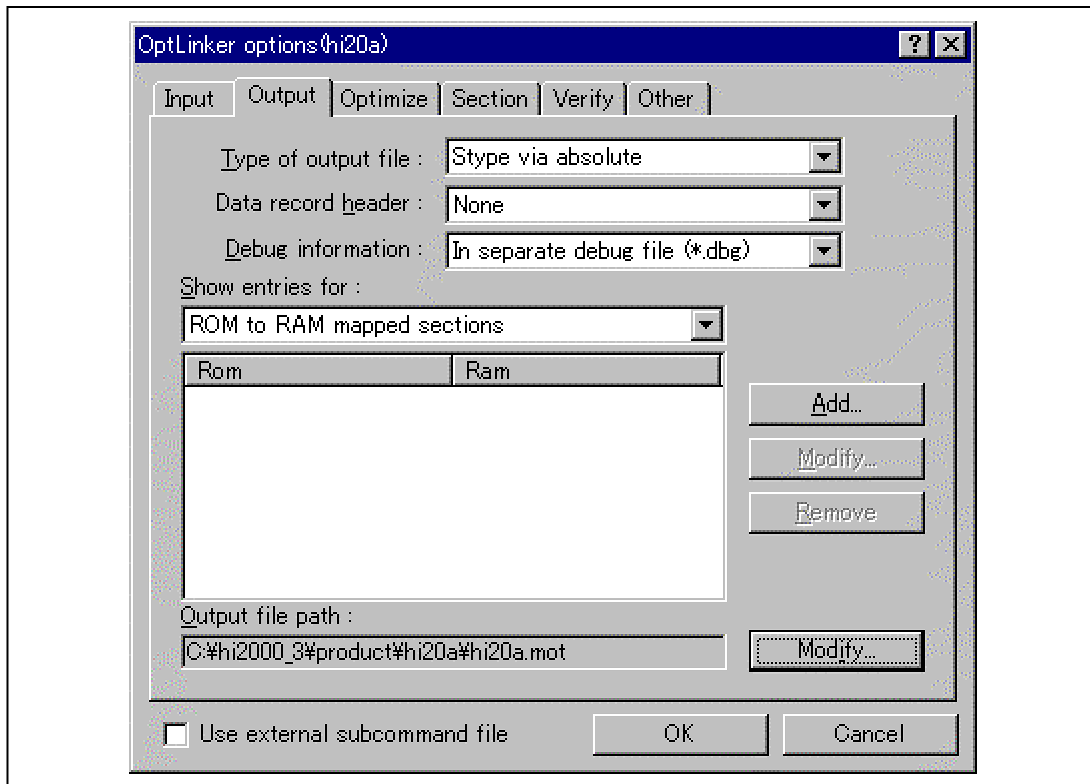


Figure 8.11 Inter-Module Optimizer Options Output Tab

3. Inter-Module Optimizer Options Section Tab

The Section tab allocates addresses to each section. The address allocations for the sections of the sample projects suit the H8S/2655 or H8S/2245. If a CPU other than H8S/2655 or H8S/2245 is used, specify the sections included in the input files and reallocate addresses to these sections.

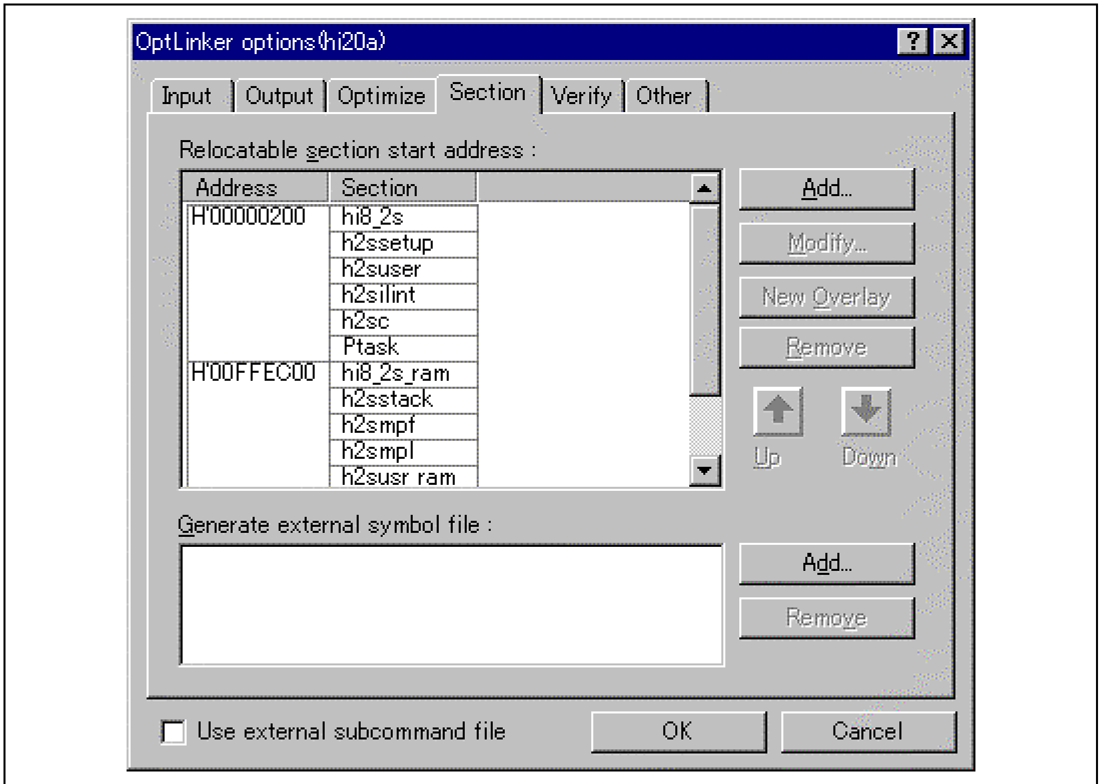


Figure 8.12 Inter-Module Optimizer Options Section Tab

The sections of the provided projects are listed in table 8.5.

Table 8.5 List of Sections Included in the Provided Project Files

Memory Type	Section	Description
ROM	h2svectr	Interrupt vector table
	hi8_2s	Kernel
	h2ssetup	Setup table
	h2suser	System initialization handler, timer initialization routine, timer interrupt handler, system termination routine, CPU initialization routine, system idling routine
	h2silint	Undefined interrupt handler
	h2sc	C-language interface library
	Ptask	Tutorial task for debugging extension (DX)
RAM	hi8_2s_ram	Kernel system work area
	h2sstack	Task stack area
	h2smpf	Fixed-size memory pool area
	h2smpl	Variable-size memory pool area
	h2susr_ram	CPU initialization routine stack area
	h2strc	Trace buffer area
	Btask	Message area of the tutorial task for debugging extension (DX)

Be sure to specify addresses for all sections in the input files. The inter-module optimizer automatically places all sections that don't have overt address specifications in sequence after the last section of the input files were specified. In such a case, the sections may not be arranged in expected order, and the program may not work properly. Specifying [Check for Unlinked Sections] in the tabbed page 'Other' will produce warning messages when there are sections that don't have address specifications. In such a case, linkage is halted. Specifying a section name that does not actually appear in the input files will also produces a warning message, but in this case the inter-module optimizer simply continues linking.

Note the following when allocating memory.

- The interrupt vector table (h2svectr) must be allocated to address H'0. When using the sample interrupt vector table, it will be automatically allocated to address H'0; therefore, the section tab does not have to be specified for the sample interrupt vector table.
- The kernel (hi8_2s) must be allocated from an even address. In the advanced mode, the section must be allocated within the range from H'xx0000 to H'xxFFFF. The upper address xx must be the same.

- The kernel system work area (hi8_2s_ram) must be allocated from an even address. In the advanced mode, the section must be allocated within the range from H'xx0000 to H'xxFFFF. The upper address xx must be the same.
- The setup table (h2ssetup) must be allocated from an even address. In the advanced mode, the section must be allocated within the range from H'xx0000 to H'xxFFFF. The upper address xx must be the same.

8.3.4 Build Execution

After application files have been added to the project and those files have been compiled, assembled, and optimized, the load module is built.

To build the load module, choose [Build] or [Build All] from the Build menu of the HEW as shown in figure 8.13.

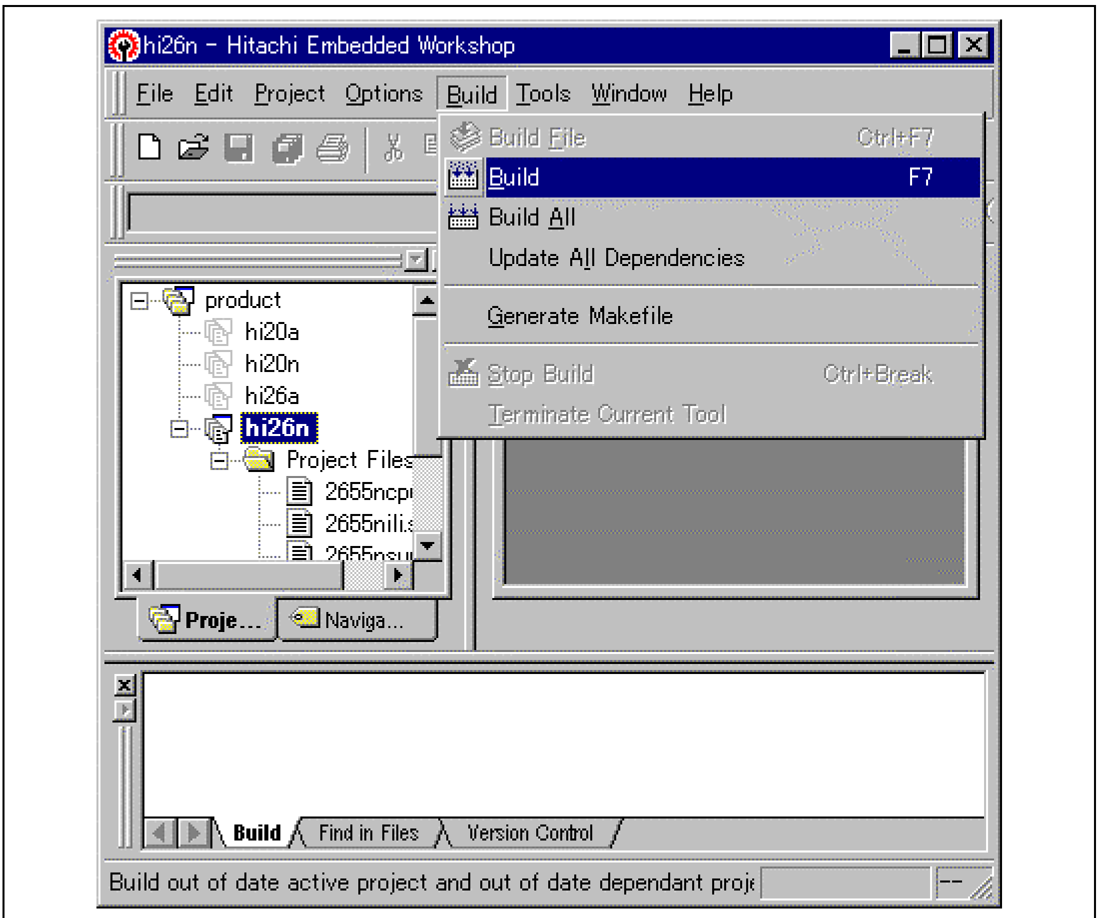


Figure 8.13 Executing the Build

8.4 C-Language Interface Library Projects

To rebuild C-language interface source files as C-language interface library files, double-click the C language interface workspace file "***xxx***cif.hws" that correspond to the target environment, as indicated in table 8.6.

Table 8.6 C-Language Interface Projects

Project Name	Description
26acif	For H8S/2600 CPU advanced mode
26ncif	For H8S/2600 CPU normal mode
20acif	For H8S/2000 CPU advanced mode
20ncif	For H8S/2000 CPU normal mode

Appendix A Memory Size

A.1 Memory Size

The memory area (RAM) size to be used by the HI2000/3 system can be calculated as follows. When calculating the stack size of the system initialization handler and timer initialization routine, use the calculation table for an interrupt handler with the same interrupt level as the kernel interrupt mask level.

A.1.1 OS Work Area Size Calculation

Calculate the OS work area size using table A.1.

Table A.1 OS Work Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
System management table (_HI_SYSMT)	10 + (4 × maximum task priority (MAXPRI))		Always necessary
Task management block (_HI_TCB)	18 × (number of tasks defined (TSKCNT))		Always necessary
Task management block 2 (_HI_TCB2)	8 × (number of tasks defined (TSKCNT))		Necessary when system calls with the timeout function are used
Event flag management block (_HI_FLGCB)	6 × (number of event flags defined (FLGCNT))		Necessary when the event flag is used
Semaphore management block (_HI_SEMCB)	6 × (number of semaphores defined (SEMCNT))		Necessary when the semaphore is used
Mailbox management block (_HI_MBXCB)	8 × (number of mailboxes defined (MBXCNT))		Necessary when the mailbox is used
Fixed-size memory pool management block (_HI_MPFCB)	6 × (number of fixed-size memory pools defined (MPFCNT))		Necessary when the fixed-size memory pool is used
Variable-size memory pool management block (_HI_MPLCB)	20 × (number of variable-size memory pools defined (MPLCNT))		Necessary when the variable-size memory pool is used

Table A.1 OS Work Area Size Calculation (cont)

Item	Calculation	Size (Bytes)	Remarks
Cyclic handler management block (_HI_CYHCB)	20 x (number of cyclic handlers defined (CYHCNT))		Necessary when the cyclic handler is used
Timer management blocks (_HI_TIMCB, _HI_TIMCB2, and _HI_TIMCB3)	10*1 + 4*2 + 14*3		
Trace buffer management block (TBACB)	8		Necessary when the trace function is used
Total			

Notes: 1. Necessary when the timer driver is used.
2. Necessary when system calls with the timeout function are used.
3. Necessary when the cyclic handler is used.

Note: If NOTUSE is selected for the timeout function definition in the setup table (label name TTMOUT), the TCB2 and TIMCB2 areas used by the timeout function are not defined.
If 0 is specified for the timer stack size in the setup table (label name TIMSTKSIZ), the timer management blocks (TIMCB, TIMCB2, and TIMCB3 areas) and timer management-related blocks (TCB2 and CYHCB) are not defined.
If 0 is specified for the trace stack size in the setup table (label name TRCSTKSIZ), the trace buffer management block is not defined.

A.1.2 OS Stack Area Size Calculation

Calculate the OS stack area size (OSSTKSIZ) using table A.2. Define the OS stack area size in the setup table.

Table A.2 OS Stack Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Stack area used by OS	18 (advanced mode) or 14 (normal mode)	18 or 14	Always necessary
Stack area for interrupts	10 × LOWINTNST*[1] + 6 × UPPINTNST*[2]		When interrupt control mode 2 or 3 is used
	8 × LOWINTNST*[1] + 4 × UPPINTNST*[2]		When interrupt control mode 0 or 1 is used
Stack area for undefined interrupts*[3]	8		When interrupt control mode 2 or 3 is used
	6		When interrupt control mode 0 or 1 is used
Total			

Notes: 1. Number of nesting interrupts of which level is equal to or lower than the kernel interrupt mask level.

2. Number of nesting interrupts (including NMIs) of which level is higher than the kernel interrupt mask level.

3. Necessary when undefined interrupts are generated.

A.1.3 Timer Interrupt Stack Area Size Calculation

Calculate the timer interrupt stack area size (TIMSTKSIZ) using table A.3.

Define the timer interrupt stack size in the setup table.

Table A.3 Timer Interrupt Stack Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Stack area used by timer interrupt handler	40 (advanced mode) or 38 (normal mode)	40 or 38	Always necessary
Stack area for interrupts	10 × LOWINTNST*[1] + 6 × UPPINTNST*[2]		When interrupt control mode 2 or 3 is used
	8 × LOWINTNST*[1] + 4 × UPPINTNST*[2]		When interrupt control mode 0 or 1 is used
Stack area for undefined interrupt*[3]	8		When interrupt control mode 2 or 3 is used
	6		When interrupt control mode 0 or 1 is used
Stack area used by cyclic handler*[4]	User-specified size		Add the size calculated by using table A.5.
Total			

Notes: 1. Number of nesting interrupts of which level is equal to or lower than the kernel interrupt mask level and higher than timer interrupt level.

2. Number of nesting interrupts (including NMIs) of which level is higher than the kernel interrupt mask level.

3. Necessary when undefined interrupts are generated.

4. When multiple cyclic handlers are used, calculate the stack size for each handler, then select the maximum size to add to the total stack size.

When a cyclic handler is written in C language, calculate the stack size from the function frame size shown in the compile listing. When issuing a system call from a cyclic handler, calculate the stack size using table A.5, Interrupt Handler Stack Area Size Calculation.

A.1.4 Task Stack Area Size Calculation

Calculate the task stack area size for each task ID using table A.4. Define the task stack area for each task ID separately in the setup table. When a task is written in C language, calculate the stack size from the function frame size shown in the compile listing. The overall size of the task stack is the sum of all the task ID sizes. When using the shared stack function, specify the maximum size used by the tasks that share the stack area.

Note: When using the shared stack function, define 8 bytes of area ranging from the end address of each stack area in the direction of ascending addresses.

Table A.4 Task Stack Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Stack area used by task	User-specified size		
Stack area used by OS	50 (H8S/2600 CPU) or 42 (H8S/2000 CPU)	50 or 42	Always necessary
Stack area for interrupts	$10 \times$ LOWINTNST*[1] + $6 \times$ UPPINTNST*[2]		When interrupt control mode 2 or 3 is used
	$8 \times$ LOWINTNST*[1] + $4 \times$ UPPINTNST*[2]		When interrupt control mode 0 or 1 is used
Stack area for system call trace	6		Necessary when trace function is used
Stack area for undefined interrupts*[3]	8		When interrupt control mode 2 or 3 is used
	6		When interrupt control mode 0 or 1 is used
Stack area for C language interface	22 (advanced mode) or 14 (normal mode)		Necessary when the task is written in C language
Stack area for shared stack function	8		Necessary when the shared stack function is used
Total			

Notes: 1. Number of nesting interrupts of which level is equal to or lower than the kernel interrupt mask level.

2. Number of nesting interrupts (including NMIs) of which level is higher than the kernel interrupt mask level.

3. Necessary when undefined interrupts are generated.

A.1.5 Interrupt Handler Stack Area Size Calculation

Calculate the stack area size for each interrupt handler using the following table A.5.

Note that the interrupt handler stack area can be shared with interrupt handlers of the same interrupt priority level. Accordingly, the maximum interrupt handler stack area size of the same interrupt priority level must be defined.

The handler stack area must be defined for each handler separately.

If an interrupt handler is written in C language, calculate the stack size from the function frame size shown in the compiler listing.

Table A.5 Interrupt Handler Stack Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Stack area used by interrupt handler	User-specified size		
Stack area used by OS	42 (advanced mode) or 38 (normal mode)	42 or 38	Always necessary
Stack area for interrupts	10 × LOWINTNST*[1] + 6 × UPPINTNST*[2]		When interrupt control mode 2 or 3 is used
	8 × LOWINTNST*[1] + 4 × UPPINTNST*[2]		When interrupt control mode 0 or 1 is used
Stack area for system call trace	6		Necessary when trace function is used
Stack area for undefined interrupts*[3]	8		When interrupt control mode 2 or 3 is used
	6		When interrupt control mode 0 or 1 is used
Stack area for C language interface	22 (advanced mode) or 14 (normal mode)		Necessary when the interrupt handler is written in C language
Total			

Notes: 1. Number of nesting interrupts of which level is equal to or lower than the kernel interrupt mask level and higher than the current interrupt level.
2. Number of nesting interrupts (including NMIs) of which level is higher than the kernel interrupt mask level.
3. Necessary when undefined interrupts are generated.

A.1.6 Fixed-Size Memory Pool Area Size Calculation

Calculate the fixed-size memory pool area size for each memory pool ID using table A.6.

The overall size of the fixed-size memory pool areas is the sum of all the memory pool ID sizes.

Define the number of the fixed-size memory blocks (MB?_CNT) and the size of the fixed-size memory block (MB?_LEN) for each memory pool ID in the setup table to reserve the memory pool area.

Table A.6 Fixed-Size Memory Pool Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Fixed-size memory pool area	Number of fixed-size memory blocks*1 × (Memory block size*2 + 4)		Management area (4 bytes) is needed for each memory block
Total			

Notes: 1. The label of the number of fixed-size memory blocks is MB?_CNT.
2. The label of the fixed-size memory block size is MB?_LEN.

A.1.7 Variable-Size Memory Pool Area Size Calculation

Calculate the variable-size memory pool area size (MPL?_SIZ) for each ID using table A.7.

The overall variable-size memory pool areas is the sum of all the memory pool ID sizes.

Define the size of the variable-size memory pool area (MPL?_SIZ) for each memory pool ID in the setup table to allocate the memory pool area.

Table A.7 Variable-Size Memory Pool Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Variable-size memory pool area (MPL?_SIZ)	Variable-size memory pool size + (16 × n*)		Management area (16 bytes) is needed for each memory block
Total			

Note: Maximum number of variable-size memory blocks acquired.

A.1.8 Trace Function Stack Area Size Calculation

Calculate the trace function stack area size (TRCSTKSIZ) using table A.8. This stack area is needed only when using the trace function.

Define the trace function stack area size in the setup table.

Table A.8 Trace Function Stack Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Stack area used by OS	26	26	Always necessary
Stack area for interrupts	6 × UPPINTNST*[1]		When interrupt control mode 2 or 3 is used
	4 × UPPINTNST*[1]		When interrupt control mode 0 or 1 is used
Stack area for undefined interrupts*[2]	8		When interrupt control mode 2 or 3 is used
	6		When interrupt control mode 0 or 1 is used
Total			

Notes: 1. Number of nesting interrupts (including NMIs) of which level is higher than the kernel interrupt mask level.

2. Necessary when undefined interrupts are generated.

A.1.9 Trace Buffer Area Size Calculation

Calculate the trace buffer area (TRC_BUF) size using table A.9. This area is needed only when using the trace function. This calculation table can be used in both normal mode and advanced mode.

Define the trace buffer area size in the setup table to reserve the trace buffer area.

Table A.9 Trace Buffer Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
Trace buffer management area	16	16	
Trace entry information area	28 × number of trace information items acquired (TRCCNT)		
Total			

A.1.10 HI2000/3 Work Area Size Calculation

Calculate the HI2000/3 work area (RAM) size using table A.10.

Table A.10 HI2000/3 Work Area Size Calculation

Item	Calculation	Size (Bytes)	Remarks
OS work area size	—		See table A.1
OS stack area size	—		See table A.2
Timer interrupt stack area size	—		See table A.3
Task stack area size (total)*1	—		See table A.4
Interrupt handler stack area size*2	—		See table A.5
Fixed-size memory pool area size (total)	—		See table A.6
Variable-size memory pool area size (total)	—		See table A.7
Trace function stack area size	—		See table A.8
Trace buffer area size	—		See table A.9
NMI interrupt handler stack area size			
System initialization handler stack area size	—		
CPU initialization routine stack area size*3	—		
Timer initialization routine stack area size	—		
Other ( )	—		
Other ( )	—		
Other ( )	—		
Other ( )	—		
Total			

Notes: 1. When the shared stack function is used, every task stack area must have an area for the shared stack management.

2. Interrupts with the same interrupt level can share the interrupt handler stack area. Accordingly, define the maximum interrupt handler stack area size that will be used by the interrupt handlers with the same level.

3. The CPU initialization routine is executed before the kernel is initiated. Accordingly, the stack area for the CPU initialization routine can be used as stack areas (RAM) other than the NMI interrupt handler stack area.

Appendix B Compiler and Assembler Options

B.1 Compiler Options

This section covers the important C compiler options used to create this system. For details on compiler options, refer to the H8S, H8/300 Series C/C++ Compiler User's Manual.

1. cpu command option

   Specifies the CPU type. Specify the appropriate value for the CPU used.

   If a program is compiled with an incorrect CPU type specified and then executed, or compiled by specifying two or more different CPU types and then executed, normal system operation cannot be guaranteed.
2. include command option

   Specifies include files.

   The kernel provides standard header file hi2000.h. The hi2000.h file is under the sample directory; include this header when required.
3. debug command option

   Specifies the addition of debugging information to the object. Specify it when using Hitachi's debugging environment.
4. list command option

   Specifies the creation of a compile list file. Important information such as stack frame size and section size is output to the list file. The list will be useful for calculating stack sizes and linking files.
5. objectfile command option

   Specifies which object module to output.

B.2 Assembler Options

This section covers the important assembler options used to create this system. For details on assembler options, refer to the H8S, H8/300 Series Cross Assembler User's Manual.

1. cpu command option

   Specifies the CPU type. Specify the appropriate value for the CPU used.

   If a program is compiled with an incorrect CPU type specified and then executed, or compiled by specifying two or more different CPU types and then executed, normal system operation cannot be guaranteed.
2. include command option

   Specifies include files.

   The kernel provides standard header file hi2000.inc. The hi2000.inc file is under the sample directory; include this header when required.

3. debug command option

   Specifies the addition of debugging information to the object. Specify it when using Hitachi's debugging environment.

4. list command option

   Specifies the creation of an assembly list file. Important information such as section sizes is output to the list file. The list will be useful for linking files.

5. objectfile command option

   Specifies which object module to output.

Appendix C Device Driver

C.1 Timer Driver

The kernel provides a sample timer driver using the timer pulse unit (TPU) and the free running timer (FRT) incorporated in the H8S series MCU. This section describes the sample timer driver. When using another hardware timer, refer to the appropriate timer hardware specifications.

A timer driver must be created and incorporated into the system when using the kernel time management function. The timer driver consists of a timer initialization routine and a timer interrupt handler.

Figure C.1 shows the timer driver processing.

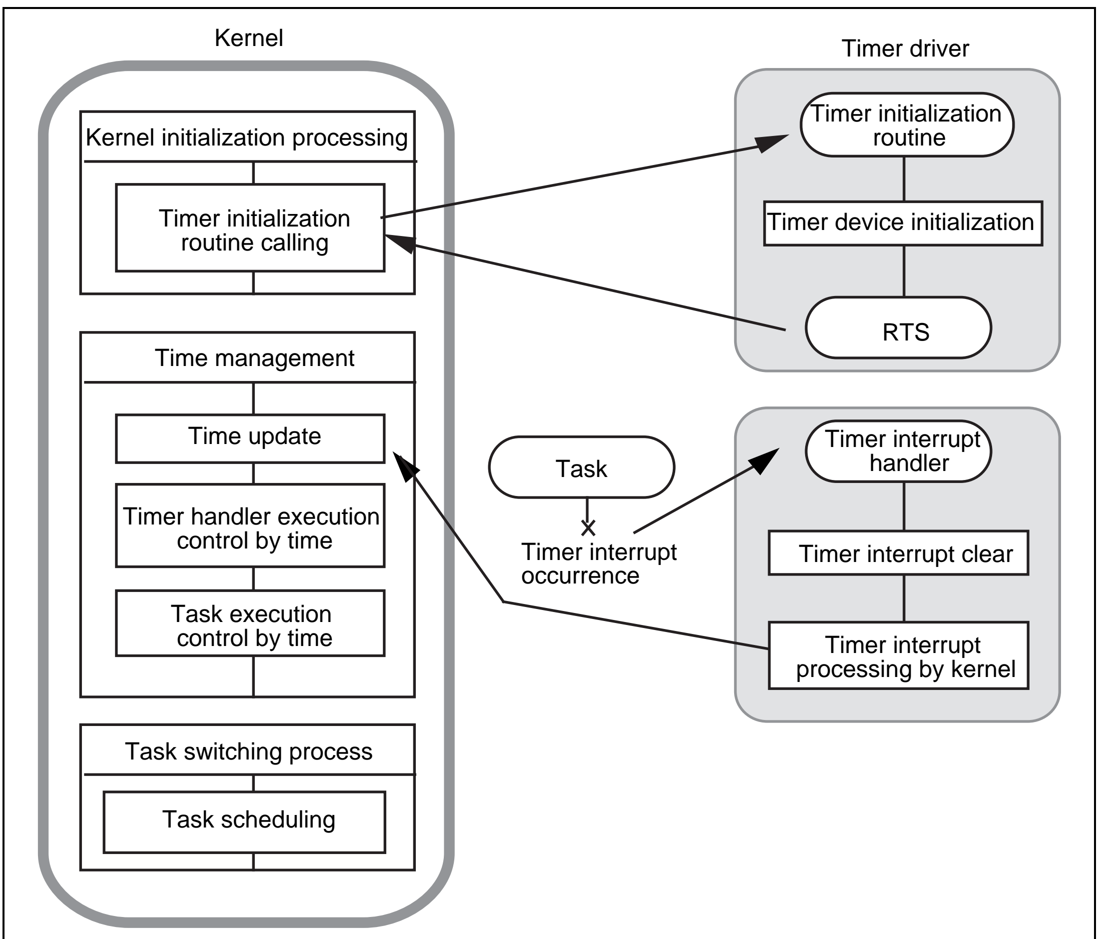


Figure C.1 Timer Driver Processing

C.1.1 Timer Initialization Routine

The timer initialization routine initializes the hardware timer to be used. Table C.1 lists the conditions for the timer initialization routine processing.

Table C.1 Conditions for Timer Initialization Routine Processing

Item	Description
Interrupt mask	Initiated in interrupt mask state.
Usable registers	The registers guaranteed in the C language programs (functions) can be used.
Stack pointer	Set the same value as that at initiation when control is returned to the kernel.
Usable system calls	No system call can be issued.
Usable stack area	When using the stack, reserve the stack area for the timer initialization routine during system configuration, and switch the stack when the timer initialization routine is initiated.
Termination	Execution is terminated by the RTS instruction. At termination, set the task state as that at initiation.

C.1.2 Timer Interrupt Handler

The timer interrupt handler is initiated by the occurrence of an interrupt from the hardware timer.

When a hardware timer interrupt occurs, the timer interrupt handler performs the timer interrupt reset processing, which clears the hardware timer interrupt, and then jumps to the kernel timer interrupt processing routine, which requests time management processing to the kernel.

The timer interrupt handler can also control task execution by issuing system calls for task-independent portion from the timer interrupt reset processing.

Table C.2 lists the conditions for the timer interrupt reset processing.

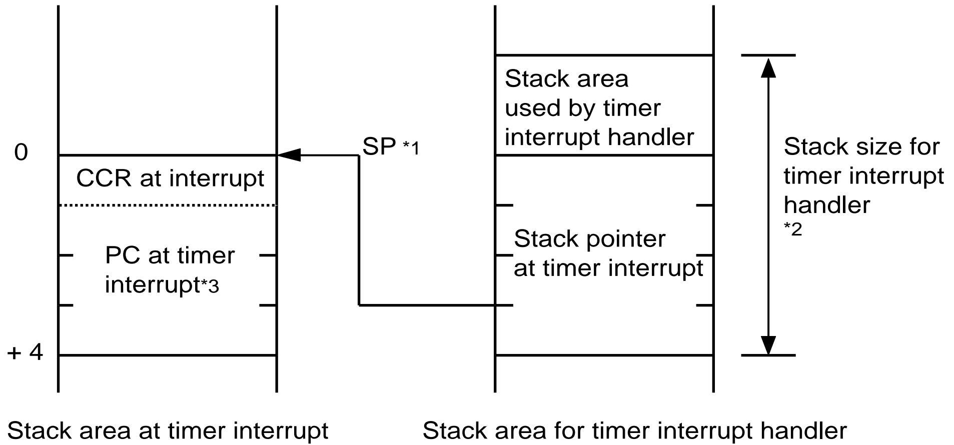
Table C.2 Conditions for Timer Interrupt Reset Processing

Item	Description
Interrupt mask	Initiated in interrupt mask state.
Usable registers	ER0 to ER6.
Stack pointer	When time management processing is not requested: When returning control to the interrupt source, set the value to that at initiation. When time management processing is requested: Set the stack pointer to the value to that when switched to timer interrupt handler. Refer to the stack state at timer interrupt reset processing termination shown in figure C.2.
Usable system calls	System calls that can be issued from task-independent portion.
Usable stack area	Reserve the stack area at system configuration and switch the stack at initiation.
Termination	Terminates processing by jumping to the kernel timer interrupt processing. jmp @_H_timsys _H_timsys is the head symbol of kernel timer interrupt processing. At termination, set the stack as that at initiation.

When the timer interrupt reset processing is terminated, set the stack pointer to the address of the timer interrupt handler stack area used when the timer interrupt is initiated, and jump to the kernel timer interrupt processing routine.

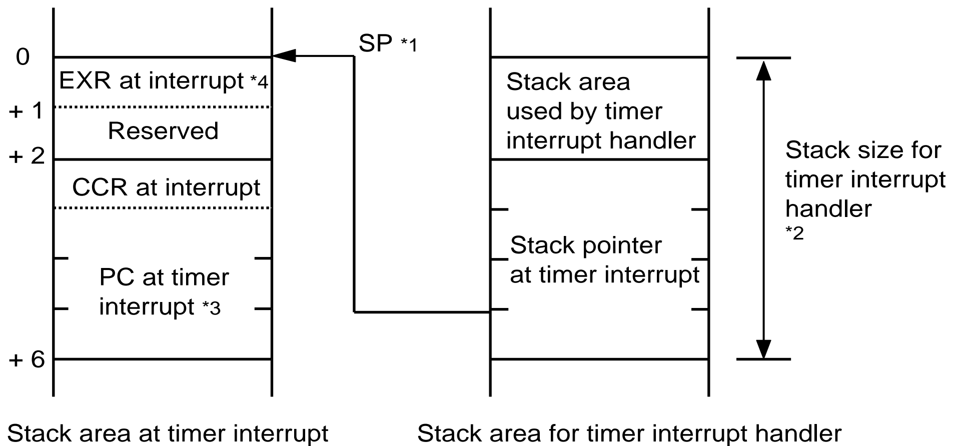
Figure C.2 shows the stack state at timer interrupt reset processing termination.

(1) Interrupt Control Mode 0 or 1



Notes: 1. Stack pointer value when execution jumps to the timer interrupt processing of the kernel.
2. For the timer interrupt stack size, refer to appendix A, Memory Size.
3. The low-order 16 bits are valid in normal mode.

(2) Interrupt Control Mode 2 or 3



Notes: 1. Stack pointer value when execution jumps to the timer interrupt processing of the kernel.
2. For the timer interrupt stack size, refer to appendix A, Memory Size.
3. The low-order 16 bits are valid in normal mode.
4. The EXR register contents are not saved in the stack in control mode 0 or 1.

Figure C.2 Stack State at Timer Interrupt Reset Processing Termination

C.1.3 Timer Driver Definition Information

A timer driver is required to use the time management function. The kernel uses the interrupts at certain intervals input from the hardware for time management. The 16-bit timer in the H8S series MCU is used as the hardware timer for time management. The timer driver consists of a timer initialization routine and a timer interrupt handler. The kernel provides an H8S series sample timer driver file sample*nnnnz*smp*nnnnz*use.src.

Timer Initialization Routine (Label Name: _HIPRG_TIMINI): Initializes the hardware timer used as a system clock. Note that the timer interrupt level must not be higher than the kernel interrupt mask level specified in the setup table.

Timer Interrupt Handler (Label Name: _H_2S_TIM): Clears the timer interrupt after a hardware timer interrupt occurs (timer interrupt reset procedure) and causes a jump to the kernel timer interrupt processing. This handler must be created if the hardware timer specifications require interrupts to be cleared.

For details on the hardware timer specifications, refer to the target MCU hardware manual.

Timer Cycle Modification: The TPU in the H8S/2655 is described as an example. The sample timer driver specifies the hardware timer cycle as 10 ms by using the TPU general register 0A (TGR0A) as an output compare register. Table C.3 shows the definition of assign directives to simplify the timer cycle modification.

Table C.3 Definition of Assign Directive for Timer Driver

Label Name	Contents	Set Value
TGRA_DATA	Data to be set in the timer general register 0A (TGR0A) Timer prescaler Selects the timer counter clock from among the following: φ φ/4 φ/16 φ/64 The timer cycle is determined by the data specified in TGR0A and the timer prescaler using the following formula. Timer cycle = x (s) Timer prescaler = n TGR0A data = x × n – 1 Example: Sample driver value CPU clock (φ) = 20 MHz Timer prescaler = φ/16 Timer cycle = 10 ms TGR0A = 0.01 (20,000,000/16) – 1 = 12,500 – 1 = H'30d3 The relationship between the timer prescaler and timer cycle range is shown below when the CPU clock φ is assumed to be 20 MHz. Timer prescaler = φ: Timer cycle range: 50.0 μs to 3.27 ms Timer prescaler = φ/4: Timer cycle range: 200.0 μs to 13.1 ms Timer prescaler = φ/16: Timer cycle range: 800.0 μs to 52.4 ms Timer prescaler = φ/64: Timer cycle range: 3200.0 μs to 209.7 ms	H'30D3
TCR_DATA	Data to be set in the timer control register 0 (TCR0) Counter clear Compares the timer counter (TCNT0) with the TGR0A, and if they match, clears the TCNT0. Timer prescaler Selects the internal clock of φ/16 as the timer counter clock.	H'22
IPRF_TPU0	The interrupt level of the TPU channel 0 interrupt handler	H'05

Timer Driver Definition and Deletion: To define the timer driver, the timer initialization routine and the timer interrupt handler must be defined. When the timer driver is not used, the timer driver information must be deleted from the interrupt vector table and the setup table, and the supplied timer driver must be deleted.

- Defining the timer initialization routine

  To define the timer initialization routine, add the label name _HIPRG_TIMINI to the head of the timer initialization routine program, and declare it with the export directive.
  To not define the timer initialization routine, set the label name _HIPRG_TIMINI to 0 by the equate directive.

- Defining the timer interrupt handler

  Define the start address of the timer interrupt handler in the interrupt vector table.

- Deleting the timer driver

  When the timer driver is not used, delete it as follows:

  — Interrupt vector table

    For the H8S/2655, delete the external reference declaration (import) from the start address of the timer interrupt handler (label name _H_2S_TIM), and define the undefined interrupt handler (label name _H_2SINT32) in vector number 32.

  — Setup table

    Modify the timer stack size (label name TIMSTKSIZ) to 0.

  — Timer driver (timer initialization routine and timer interrupt handler)

    Remove the timer initialization routine program (label name _HIPRG_TIMINI), and set the label name _HIPRG_TIMINI to 0 by the equate directive. Remove the timer interrupt handler program (label name _H_2S_TIM). Remove the external reference symbol declaration (import) from the OS timer interrupt processing (label name _H_timsys).

Precautions on Using a Timer other than those of the H8S Series MCU: When a timer other than the H8S series MCU's TPU and FRT is used, a new timer driver must be created.

Appendix D Error Codes

D.1 System Call Error Codes

Table D.1 System Call Error Codes

Error Code (Mnemonic)		Error Code (ercd)		Error Check Type	Error Contents
1	E_OK	H'0000	(H'0)	[k]	Normal termination
2	E_RSFN	H'ffec	(–H'14)	[p]	Reserved function code number (Undefined function code specified)
3	E_PAR	H'ffdf	(–H'21)	[p]/[k]	Parameter error
4	E_ID	H'ffdd	(–H'23)	[p]	Invalid ID number
5	E_NOEXS	H'ffcc	(–H'34)	[p]	Object does not exist Object is undefined
6	E_OBJ	H'ffc1	(–H'3f)	[k]	Object state is illegal
7	E_CTX	H'ffbb	(–H'45)	[p]/[k]	Context error
8	E_QOVR	H'ffb7	(–H'49)	[k]	Overflow of queuing or nesting
9	E_TMOUT	H'ffab	(–H'55)	[k]	Polling failure or timeout
10	E_RLWAI	H'ffaa	(–H'56)	[k]	Wait state has been released forcibly
11	EV_ILBLK	H'ff1e	(–H'e2)	[k]	Returns illegal memory block

Error check type: [p] is an error that is checked when the parameter checking function is incorporated.

[k] is an error that is checked even when the parameter checking function is not incorporated.

D.2 Debugging Extension Errors

Table D.2 Debugging Extension Error Messages

Error Message	Meaning and Actions to Take
Cannot open HIOS window - no HIOS program loaded.	The load module is not loaded. Load the load module.
Cannot open memory display window @ H'xxxxxx Operation not implemented on this version of HDI.	The Memory window cannot be displayed. This window is not supported by this version of HDI.
Cannot open program code window @ H'xxxxxx Operation not implemented on this version of HDI.	The Program window cannot be displayed. This window is not supported by this version of HDI.
ERROR : Command Already On Stack.	The specified command request has already requested.
ERROR : Demon Code Not Present. Command Cancelled.	The debug daemon is not installed. Install the debug daemon referring to Installing the Debug Daemon in section 4.4.
ERROR : Demon Code Not Running. Command Cancelled.	The debug daemon is not initialized. Execute Go Reset for the kernel and initialize the debug daemon.
Error : Number Out of Range	The data has exceeded the specifiable range. Check the specified data.
Error: Invalid input expression	The specified data is invalid. Check the specified data.
HIOS Error H'xxxx : <Error Message>	An error has occurred for the debug daemon system call. Check the state of the specified ID.
Invalid Expression	Not a specifiable flag value. Check the value of the specified flag.
Invalid Format in Message Address	The format of the message address is invalid. Check the message address.
Invalid Format in Message String!	The specified message string format is invalid. Check the specified message.
Timer Value invalid or wrong format!	The timer value or the format is invalid. Check the specified timer value.
Unable to remove message from selected Mailbox.	Cannot delete the message in the selected mailbox. Check the selected mailbox.

Table D.2 Debugging Extension Error Messages (cont)

Error Message	Meaning and Actions to Take
Unable to set breakpoint on HDI!	Breakpoints cannot be specified since the task is in the ROM area. Check the specified area.
Value Too Large	The flag value has exceeded the specifiable range. Check the specified flag value.

Appendix E System Call Function Codes

E.1 System Call Function Codes

The following table lists the system calls and their function codes for the system call trace function.

Table E.1 System Calls and Function Codes

No.	System Call	Function Code	No	System Call	Function Code
1	ista_tsk	H'ff09 (-H'f7)	21	ref_cyc	H'ffa4 (-H'5c)
2	trcv_msg	H'ff54 (-H'ac)	22	get_tim	H'ffac (-H'54)
3	twai_sem	H'ff55 (-H'ab)	23	set_tim	H'ffad (-H'53)
4	twai_flg	H'ff56 (-H'aa)	24	rel_blf	H'ffb1 (-H'4f)
5	tget_blk	H'ff58 (-H'a8)	25	get_blf	H'ffb3 (-H'4d)
6	tget_blf	H'ff59 (-H'a7)	26	ref_mpf	H'ffb4 (-H'4c)
7	rel_blk	H'ff71 (-H'8f)	27	ret_int	H'ffbb (-H'45)
8	get_blk	H'ff73 (-H'8d)	28	ref_ims	H'ffbc (-H'44)
9	ref_mpl	H'ff74 (-H'8c)	29	chg_ims	H'ffbd (-H'43)
10	isnd_msg	H'ff84 (-H'7c)	30	snd_msg	H'ffc1 (-H'3f)
11	isig_sem	H'ff85 (-H'7b)	31	rcv_msg	H'ffc3 (-H'3d)
12	iset_flg	H'ff86 (-H'7a)	32	ref_mbx	H'ffc4 (-H'3c)
13	iwup_tsk	H'ff87 (-H'79)	33	sig_sem	H'ffc9 (-H'37)
14	irot_rdq	H'ff8a (-H'76)	34	wai_sem	H'ffcb (-H'35)
15	prcv_msg	H'ff94 (-H'6c)	35	ref_sem	H'ffcc (-H'34)
16	preq_sem	H'ff95 (-H'6b)	36	set_flg	H'ffd0 (-H'30)
17	pol_flg	H'ff96 (-H'6a)	37	clr_flg	H'ffd1 (-H'2f)
18	pget_blk	H'ff98 (-H'68)	38	wai_flg	H'ffd2 (-H'2e)
19	pget_blf	H'ff99 (-H'67)	39	ref_flg	H'ffd4 (-H'2c)
20	act_cyc	H'ffa2 (-H'5e)	40	can_wup	H'ffd8 (-H'28)

Table E.1 System Calls and Function Codes (cont)

No.	System Call	Function Code	No	System Call	Function Code
41	wup_tsk	H'ffd9 (-H'27)	50	chg_pri	H'ffe5 (-H'1b)
42	slp_tsk	H'ffda (-H'26)	51	ter_tsk	H'ffe7 (-H'19)
43	tslp_tsk	H'ffdb (-H'25)	52	get_tid	H'ffe8 (-H'18)
44	rsm_tsk	H'ffdd (-H'23)	53	sta_tsk	H'ffe9 (-H'17)
45	sus_tsk	H'ffdf (-H'21)	54	ext_tsk	H'ffeb (-H'15)
46	rel_wai	H'ffe1 (-H'1f)	55	ref_tsk	H'ffec (-H'14)
47	dis_dsp	H'ffe2 (-H'1e)	56	get_ver	H'fff0 (-H'10)
48	ena_dsp	H'ffe3 (-H'1d)	57	loc_cpu	H'fff8 (-H'8)
49	rot_rdq	H'ffe4 (-H'1c)	58	unl_cpu	H'fff9 (-H'7)