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.

User's Manual

RENESAS

M3T-MR308 V.1.20

Reference Manual

Real-time OS for M32C/80,M16C/80 Series

Renesas Electronics
www.renesas.com

Rev.1.00 2003.09

Active X, Microsoft, MS-DOS, Visual Basic, Visual C++, Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.
Sun, Solaris, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries, and are used under license.
Linux is a trademark of Linus Torvalds.
Turbolinux and its logo are trademarks of Turbolinux, Inc.
IBM and AT are registered trademarks of International Business Machines Corporation.
Intel and Pentium are registered trademarks of Intel Corporation.
Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.
All other brand and product names are trademarks, registered trademarks or service marks of their respective holders.

Keep safety first in your circuit designs!

- Renesas Technology Corporation and Renesas Solutions Corporation put 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

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology 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, Renesas Solutions Corporation or a third party.
- Renesas Technology Corporation and Renesas Solutions Corporation assume 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.
- 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 and Renesas Solutions Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology 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 and Renesas Solutions Corporation assume 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 and Renesas Solutions Corporation by various means, including the Renesas home page (http://www.renesas.com).
- 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 and Renesas Solutions Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology 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, Renesas Solutions Corporation or an authorized Renesas Technology 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.
- The prior written approval of Renesas Technology Corporation and Renesas Solutions Corporation is necessary to reprint or reproduce in whole or in part these materials.
- 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.
- Please contact Renesas Technology Corporation or Renesas Solutions Corporation for further details on these materials or the products contained therein.

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

\SUPPORT\Product-name\SUPPORT.TXT

Renesas Tools Homepage http://www.renesas.com/en/tools

Preface

The MR308 is a real-time operating system[1] for the M16C/80 microcomputers. The MR308 conforms to the μITRON Specification.[2]

This manual describes the function and configuration.

Right of Software Use

The right of software use conforms to the software license agreement.

You can use the MR308 for your product development purposes only, and are not allowed to use it for the other purposes.

You should also note that this manual does not guarantee or permit the exercise of the right of software use.

Document List

The following sets of documents are supplied with the MR308.

- Release Note

  Presents a software overview and describes the corrections to the Users Manual and Reference Manual.

- Users Manual (PDF file)

  Describes the procedures and precautions to observe when using the MR308 for programming purposes.

- Reference Manual (PDF file)

  Describes the MR308 system call procedures and typical usage examples. Before reading the Users Manual, be sure to read the Release Note.

  Please read the release note before reading this manual.

---

[1] Hereinafter abbreviated "real-time OS"

[2] The μITRON Specification is originated by Dr.Ken Sakamura and his laboratory members at the Faculty Science of University of Tokyo. Therefore,Dr.Ken Sakamura holds the copyright on the μITRON Specification. By his consent,the MR308 is produced in compilance with the μITRON Specification.

Contents

Chapter 1 Interpreting the System Call Reference 1

1.1. Interpreting the System Call Reference 2

1.2. Performance for Individual System Calls 4

1.3. Processing Time for the Scheduler 6

1.3.1. Processing Time for the Scheduler Activated by a System Call 6
1.3.2. Processing Time for the Scheduler Activated by an Interrupt Handler 6
1.3.3. Processing Time for the System Timer 6

1.4. Necessary Stack Size 7

1.5. Stack Size Calculation Method 9

1.5.1. User Stack Calculation Method 11
1.5.2. System Stack Calculation Method 13

Chapter 2 System Call Reference 17

2.1. Task Management Functions 18

2.1.1. sta_tsk(Start Task) 18
2.1.2. ista_tsk(Start Task) 21
2.1.3. ext_tsk(Exit Task) 23
2.1.4. ter_tsk(Terminate Task) 25
2.1.5. dis_dsp(Disable Dispatch) 27
2.1.6. ena_dsp(Enable Dispatch) 29
2.1.7. chg_pri(Change Task Priority) 31
2.1.8. ichg_pri(Change Task Priority) 33
2.1.9. rot_rdq(Rotate Ready Queue) 35
2.1.10. irot_rdq(Rotate Ready Queue) 38
2.1.11. rel_wai(Release Task Wait) 40
2.1.12. irel_wai(Release Task Wait) 42
2.1.13. get_tid(Get Self Task ID) 44
2.1.14. ref_tsk(Refer Task Status) 46

2.2. Synchronization Functions Attached to Task 49

2.2.1. sus_tsk(Suspend Task) 49
2.2.2. isus_tsk(Suspend Task) 51
2.2.3. rsm_tsk(Resume Task) 53
2.2.4. irsm_tsk(Resume Task) 55
2.2.5. slp_tsk(Sleep Task) 57
2.2.6. tslp_tsk(Sleep Task with Timeout) 59
2.2.7. wup_tsk(Wakeup Task) 61
2.2.8. iwup_tsk(Wakeup Task) 63
2.2.9. can_wup(Cancel Wakeup Task) 65

2.3. Eventflags 67

2.3.1. set_flg(Set Eventflag) 67

2.3.2. iset_flg(Set Eventflag) 69
2.3.3. clr_flg(Clear Eventflag) 71
2.3.4. wai_flg(Wait Eventflag) 73
2.3.5. twai_flg(Wait Eventflag with Timeout) 76
2.3.6. pol_flg(Poll Eventflag) 78
2.3.7. ref_flg(Refer Eventflag Status) 80

2.4. Semaphore 82

2.4.1. sig_sem(Signal Semaphore) 82
2.4.2. isig_sem(Signal Semaphore) 84
2.4.3. wai_sem(Wait on Semaphore) 86
2.4.4. twai_sem(Wait on Semaphore with Timeout) 88
2.4.5. preq_sem(Poll and Request Semaphore) 90
2.4.6. ref_sem(Refer Semaphore Status) 92

2.5. Mailbox 94

2.5.1. snd_msg(Send Message to Mailbox) 94
2.5.2. isnd_msg(Send Message to Mailbox) 97
2.5.3. rcv_msg(Receive Message from Mailbox) 99
2.5.4. trcv_msg(Receive Message with Timeout) 102
2.5.5. prcv_msg(Poll and Receive Message) 104
2.5.6. ref_mbx(Refer Mailbox Status) 106

2.6. Interrupt Management Function 108

2.6.1. ret_int(Return from Interrupt Handler) 108
2.6.2. loc_cpu(Lock CPU) 110
2.6.3. unl_cpu(Unlock CPU) 112

2.7. Memorypool Management Function 114

2.7.1. pget_blf(Poll and Get Fixed-size Memory Block) 114
2.7.2. rel_blf(Release Fixed-size Memory Block) 116
2.7.3. ref_mpf(Refer Fixed-size Memorypool Status) 118
2.7.4. pget_blk(Poll and Get Variable-size Memory Block) 120
2.7.5. rel_blk(Release Variable-size Memory Block) 122
2.7.6. ref_mpl(Refer Variable-size Memorypool Status) 124

2.8. Time Management Function 126

2.8.1. set_tim(Set Time) 126
2.8.2. get_tim(Get Time) 128
2.8.3. dly_tsk(Delay Task) 130
2.8.4. act_cyc (Activate Cyclic Handler) 132
2.8.5. ref_cyc(Refer Cyclic Handler Status) 134
2.8.6. ref_alm(Refer Alarm Handler Status) 136

2.9. Version Management Function 138

2.9.1. get_ver(Get Version Information) 138

2.10. Implementation-Dependent System Call 140

2.10.1. vrst_msg(Reset Message) 140
2.10.2. vrst_blf(Reset Fixed-Memory Block) 142
2.10.3. vrst_blk(Reset Variable-Memory Block) 144

Chapter 3 Appendix 147

3.1. List of System calls 148

3.2. List of Error code 150

3.3. Assembly Language Interface 151

3.4. C Language Interface 155

3.5. **Data Type** 157

3.6. **Common Constants and Packet Format of Structure** 158

Index 161

Chapter 1 Interpreting the System Call Reference

1.1. Interpreting the System Call Reference

The system call reference is written in the following format:

[(System call name)]

System call name → the function of the system call

[(Calling by the assembly language)]

```
.include mr308.inc
Calling by the assembly language
```

<< Argument >>

```
Explanation of system call parameters

Parameters are written as macro arguments.
Argment name     Size    Explanation
The size is indicated by the following symbols:
[–––*]    1-byte  data
[––**]    2-byte  data
[–***]    3-byte  data (address data)
[****]    4-byte  data
The registers that can be used as the arguments of 1- byte data are R0H/R0L
and R1H/R1L only.
```

<< Register setting >>

A value is shown that is set the register after issuing a system call macro.

Register name	Register content after issuing system call
*1	*2

*1 Register name. Written in this column are R0,R1,R2,R3,A0.

*2 Indicates the content that is set in each register. Description '--' means that the content is saved if the register is set to be used, and that the content is indeterminate if the register is not set to be used.

FLG is such that the values of IPL, U, I, and B before a system call are saved; other flags are indeterminate.

The same applies for registers not listed in this table (e.g., R3, A1, SB, and FB). Namely, the content is saved if the register is set to be used, or the content is indeterminate if the register is not set to be used.

The registers used by each (return) parameter are approximately predetermined as follows:

R0 register(16bits)	Function code (set in system call macro) and error code
R1 register(16bits)	wfmode (wai_flg and pol_flg system call wait mode) and lower 16 bits of packet address
R2 register(16bits)	lower 16 bits of packet address and other parameters
R3 register(16bits)	upper 16 bits of packet address and other parameters
A0 register(24bits)	ID number of object
A1 register(24bits)	24 bits od packet address

[(Calling by the C language)]

Calling an MR308 function from the C language

<< Argument >>

Declaration of argument type

<< Return value >>

Description of the return value resulted from a call

Note that the types used in the system call reference are defined in the include file "mr308.h" The definitions are as Appendix 3.5

[(Error codes)]

[Error code name] [error code value]: [the meaning of error code]
Error code character strings such as E_OK are defined in "mr308.h" by using "#define" and in "mr308.inc" by using ".EQU" To determine errors, use these defined character strings.[3]

[(Function description)]

Detail functional description

[(Usage example)]

Usage example

[3] If an error code value is directly written, the compatibility with the future versions is not assured.

1.2. Performance for Individual System Calls

The processing time (except processing time to pass arguments to the systemcall) and the maximum interrupt disable time for individual system calls are given below.

The maximum interrupt disable time is with respect to the OS-dependent interrupt (the interrupt that issues a system call). The maximum interrupt disable time with respect to the OS-independent interrupt (the interrupt that doesn't issue a system call) is

1μs

The processing time is the result of measurement by use of a 20-MHz operating frequency and a 16-bit bus.

It is in μs. α stands for the scheduler's processing time.

System call	Processing time	Maximum interrupt disable time	System call	Processing time	Maximum interrupt disable time
sta_tsk	11+α	13	isig_sem	14	14
ista_tsk	11	11	wai_sem	9+α	11
ext_tsk	4+α	4	preq_sem	6	6
ter_tsk	16+α	17	ref_sem	6	6
dis_dsp	3	3	snd_msg	15+α	16
ena_dsp	9+α	9	isnd_msg	15	15
chg_pri	11+α	12	rcv_msg	9+α	11
ichg_pri	8	8	prcv_msg	8	8
rot_rdq	8+α	9	ref_mbx	7	7
irot_rdq	7	7	loc_cpu	4	-
rel_wai	18+α	19	unl_cpu	7+α	-
irel_wai	18	18	set_tim	6	6
get_tid	6	6	get_tim	6	6
ref_tsk	10	10	pget_blf	15	14
sus_tsk	8+α	9	rel_blf	9	9
isus_tsk	7	7	ref_mpf	15	15
rsm_tsk	8+α	9	pget_blk	29+α	6
irsm_tsk	8	8	rel_blk	more than 16+α	6
slp_tsk	7+α	8			
wup_tsk	11+α	13	ref_mpl	11	11
iwup_tsk	11	11	act_cyc	6	6
can_wup	7	6	ref_cyc	6	6
wai_flg	10+α	12	ref_alm	9	9
clr_flg	5	5	get_ver	11	11
pol_flg	8	6	vrst_msg	6	6
ref_flg	6	6	vrst_blf	5	5
sig_sem	14+α	16	vrst_blk	35	4

System call	β	Processing time	Maximum interrupt disable time
dly_tsk	0	$7+\alpha$	13
	more than 1	$11+4(\beta-1)+\alpha$	
tslp_tsk	0	$10+\alpha$	14
	more than 1	$12+4(\beta-1)+\alpha$	
twai_flg	0	$12+\alpha$	17
	more than 1	$12+4(\beta-1)+\alpha$	
twai_sem	0	$6+\alpha$	16
	more than 1	$11+4(\beta-1)+\alpha$	
trcv_msg	0	$8+\alpha$	16
	more than 1	$10+4(\beta-1)+\alpha$	

β: The number of tasks that retrieved the timeout queue.

System call	β	Processing time	Maximum interrupt disable time
set_flg	0	$6+4\gamma+\alpha$	20
	more than 1	$18+15(\beta-1)+4\gamma+\alpha$	
iset_flg	0	$7+4\gamma$	18
	more than 1	$20+15(\beta-1)+4\gamma$	

β: The number of tasks that satisfied the wait condition.

γ: The number of tasks that didn't satisfy the wait condition.

See Interrupt Handler and Schedular Processing Times on page 6 for details of the ret_int processing time and interrupt inhibit time.

1.3. Processing Time for the Scheduler

1.3.1. Processing Time for the Scheduler Activated by a System Call

The processing time and the interrupt disable time for the scheduler activated by a system call from a task are given below.

- Processing time
  5+0.7(x-1) µs
  x: Priority

- Interrupt disable time
  5+0.7(x-1) µs
  x: Priority (Values of 32 and greater are processed as 32)

1.3.2. Processing Time for the Scheduler Activated by an Interrupt Handler

The processing time and interrupt disable time for the scheduler activated at the time of returning from an interrupt are given below.

- Processing time
  10+0.7(x-1) µs
  x: Priority

- Interrupt disable time
  6+0.6(x-1) µs
  x: Priority(Values of 32 and greater are processed as 32)

1.3.3. Processing Time for the System Timer

- Processing time
  Processing time the system timer
  = 5 + (processing time for system time update +)
  (processing time for a cyclic handler +)
  (processing time for an alarm handler +)
  processing time for timeout rising +
  processing time for ret_int µs

  Processing time for the system timer = 1µs
  Processing time for a cyclic activation handler
  = 2× n + Σ (1.1+ hander executiontime)
  Processing time for an alarm handler
  = 2+ Σ (4× n + hander executiontime)
  n: the number of registered handlers
  Processing time for timeout rising = 3 (Case of no task in the timer queue)
  9+12(n–1)
  n: the number of tasks that rise

- Maximum interrupt disable time
  12µs

1.4. Necessary Stack Size

Table 1.1 lists the stack sizes (system stack) used by system calls that can be issued from tasks.

Table 1.1 Stack Sizes Used by System Calls Issued from Tasks (in bytes)

System call	Stack size		System call	Stack Size	
	User Stack	System Stack		User Stack	System Stack
sta_tsk	0	4	twai_flg	0 (6*)	8
ext_tsk	0	4	sig_sem	0	8
ter_tsk	0	8	wai_sem	0	4
dis_dsp	0	0	twai_sem	0	8
ena_dsp	0	0	snd_msg	0	8
chg_pri	0	4	rcv_msg	0 (8*)	4
rot_rdq	0	0	trcv_msg	0 (8*)	8
rel_wai	0	8	loc_cpu	0	0
sus_tsk	0	4	unl_cpu	0	0
rsm_tsk	0	4	dly_tsk	0	8
slp_tsk	0	4	pget_blk	0 (8*)	34
tslp_tsk	0	8	rel_blk	0	62
wup_tsk	0	8	vrst_msg*	12	0
set_flg	0	8	vrst_blf*	12	0
wai_flg	0 (6*)	4	vrst_blk*	50	0

*: Stack sizes used by system call in C programs.

Table 1.2 lists the stack sizes (system stack) used by system calls that can be issued from handlers.

Table 1.2 Stack Sizes Used by System Calls Issued from Handlers (in bytes)

System call	Stack Size	System call	Stack Size
ista_tsk	16	iwup_tsk	24
ichg_pri	16	iset_flg	28
irot_rdq	16	isig_sem	24
irel_wai	20	isnd_msg	28
isus_tsk	16	ret_int	12
irsm_tsk	16		

Table 1.3 lists the stack sizes (system stack) used by system calls that can be issued from both tasks and handlers. If the system call issued from task, system uses user stack. If the system call issued from handler, system uses system stack.

Table 1.3 Stack Sizes Used by System Calls Issued from Tasks and Handlers (in bytes)

System call	Stack Size	System call	Stack Size
get_tid	12 (20*)	act_cyc	12
can_wup	12 (18*)	get_ver	16
clr_flg	12	ref_tsk	20
pol_flg	12 (18*)	ref_flg	12
preq_sem	12	ref_sem	12
prcv_msg	16 (24*)	ref_mbx	12
pget_blf	12 (24*)	ref_mpf	16
rel_blf	20	ref_mpl	12
set_tim	12	ref_cyc	12
get_tim	12	ref_alm	12

*: Stack sizes used by system call in C programs.

1.5. Stack Size Calculation Method

The MR308 provides two kinds of stacks: the system stack and the user stack. The stack size calculation method differ between the stacks.

- User stack

  This stack is provided for each task. Therefore, writing an application by using the MR308 requires to allocate the stack area for each stack.

- System stack

  This stack is used inside the MR308 or during the execution of the handler.

  When a task issues an system call, the MR308 switches the user stack to the system stack. (See Figure 1.1)

  The system stack uses interrupt stack(ISP).

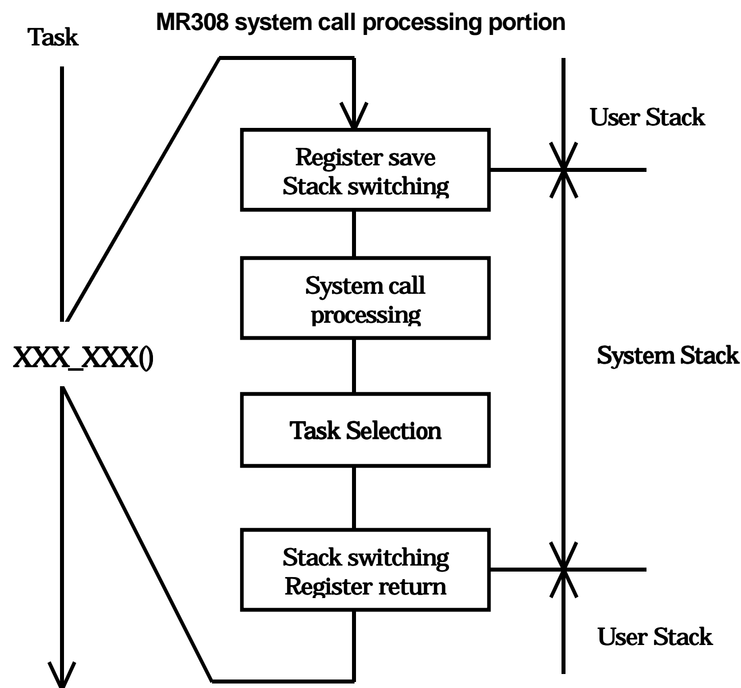

Figure 1.1: System Stack and User Stack

The system stack and the user stack for each task are allocated by the stack section in memory.

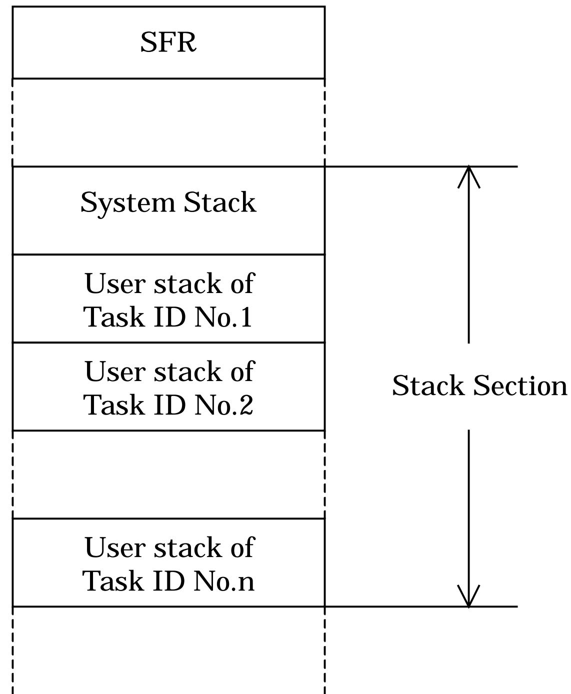

Figure 1.2: Layout of Stacks

1.5.1. User Stack Calculation Method

User stacks must be calculated for each task. The following shows an example for calculating user stacks in cases when an application is written in the C language and when an application is written in the assembly language.

- When an application is written in the C language

  Using the stack size calculation utility STK Viewer[4], calculate the stack size of each task. The necessary stack size of a task is the sum of the stack size output by STK Viewer plus a context storage area of 30 bytes[5] The following shows how to calculate a stack size using

- When an application is written in the assembly language

  - **Sections used in user program**
    The necessary stack size of a task is the sum of the stack size used by the task in subroutine call plus the size used to save registers to a stack in that task.

  - **Sections used in MR308**
    The sections used in MR308 refer to a stack size that is used for the system calls issued.

    MR308 requires that if you issue only the system calls that can be issued from tasks, 6 bytes of area be allocated for storing the PC and FLG registers. Also, if you issue the system calls that can be issued from both tasks and handlers, see the stack sizes listed in Table 1.2 to ensure that the necessary stack area is allocated.

    Furthermore, when issuing multiple system calls, include the maximum value of the stack sizes used by those system calls as the sections used by MR308 as you calculate the necessary stack size.

Therefore,

User stack size =

Sections used in user program + **registers used** + **Sections used in MR308**

(registers used is total size of used registers. If you used R0,R1,R2 and R3, add by 2bytes. If you used A0,A1,SB and FB, add by 4bytes.)

Figure 1.3 shows an example for calculating a user stack. In the example below, the registers used by the task are R0, R1, and A0.

[4] STK Viewer is a utility to calculate the stack size included with Renesas C Compiler NC308WA.
[5] If written in the C language, this size is fixed.

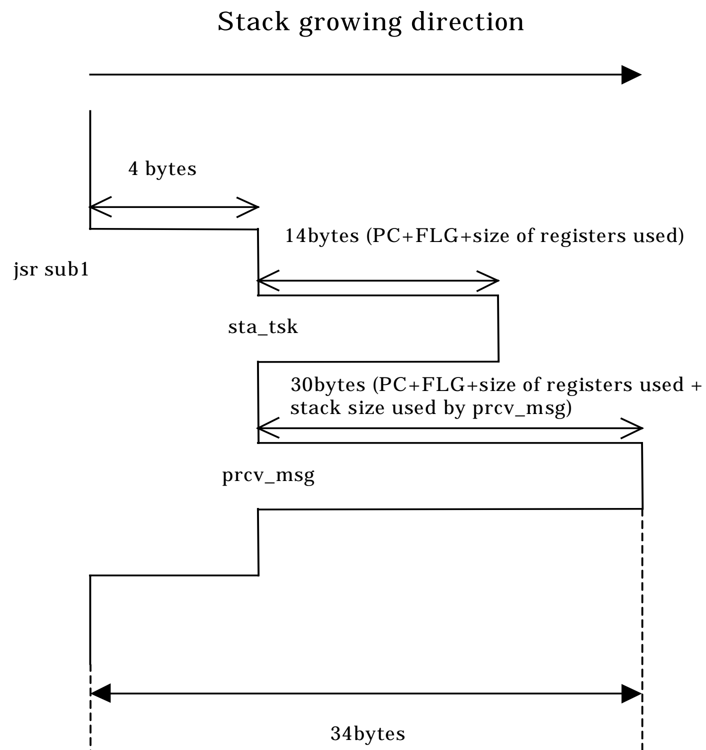

Figure 1.3: Example of User Stack Size Calculation

1.5.2. System Stack Calculation Method

The system stack is most often consumed when an interrupt occurs during system call processing followed by the occurrence of multiple interrupts.[6] The necessary size (the maximum size) of the system stack can be obtained from the following relation:

$$\text{Necessary size of the system stack} = \alpha + \Sigma\beta i(+ \gamma)$$

● α

The maximum system stack size among the system calls to be used.7

When sta_tsk, ext_tsk, and dly_tsk are used for example, according to the Table 1.1 ,each of system stack size is the following.

System call	System Stack Size
sta_tsk	4bytes
ext_tsk	4bytes
slp_tsk	4bytes
dly_tsk	8bytes

Therefore,the maximum system stack size among the system calls to be used is the 8 bytes of dly_tsk.

● βi

The stack size to be used by the interrupt handler.[8] The details will be described later.

● γ

Stack size used by the system clock interrupt handler. This is detailed later.

[6] After switchover from user stack to system stack
[7] Refer Section 1.4 for the system stack size used for each individual system call.
[8] OS-dependent interrupt handler (not including the system clock interrupt handler here) and OS-independent interrupt handler.

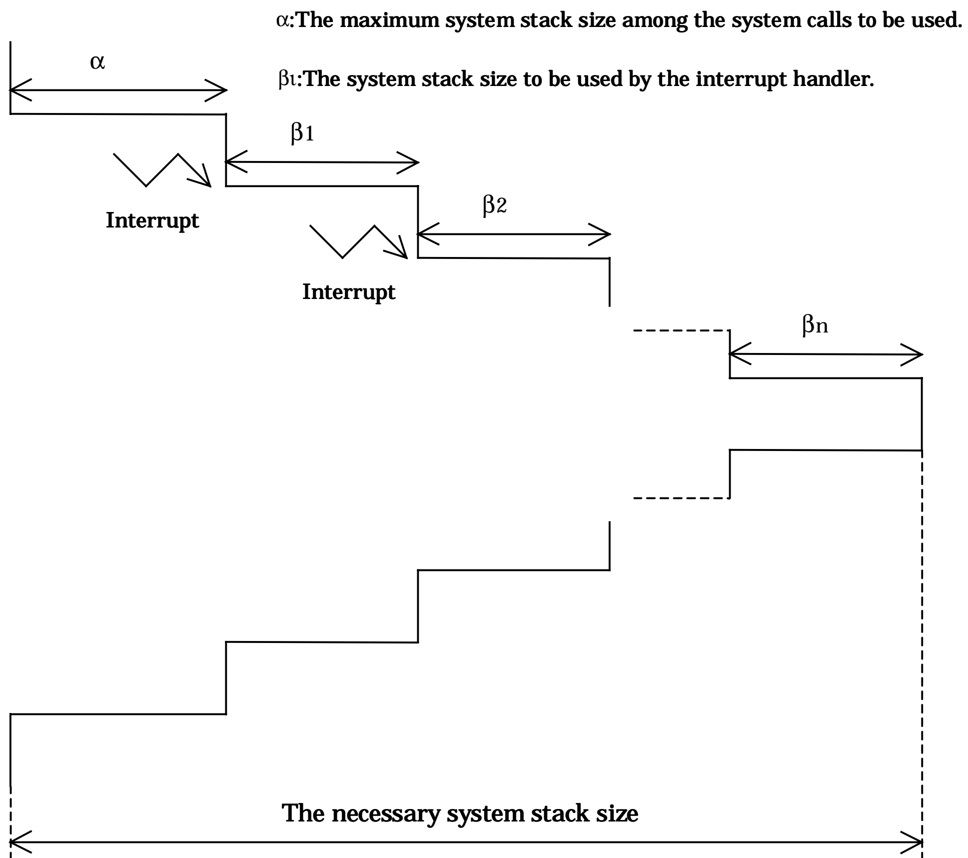

Figure 1.4:System Stack Calculation Method

[(Stack size βi used by interrupt handlers)]

The stack size used by an interrupt handler that is invoked during a system call can be calculated by the equation below.

The stack size βi used by an interrupt handler is shown below.

- **C language**
  Using the stack size calculation utility STK Viewer[9], calculate the stack size of each interrupt handler.
- **Assembly language**
  The stack size to be used by OS-dependent interrupt handler

  = **register to be used** + **user size** + **stack size to be used by system call**

  The stack size to be used by OS-independent interrupt handler

  = **register to be used** + **user size**

User size is the stack size of the area written by user.

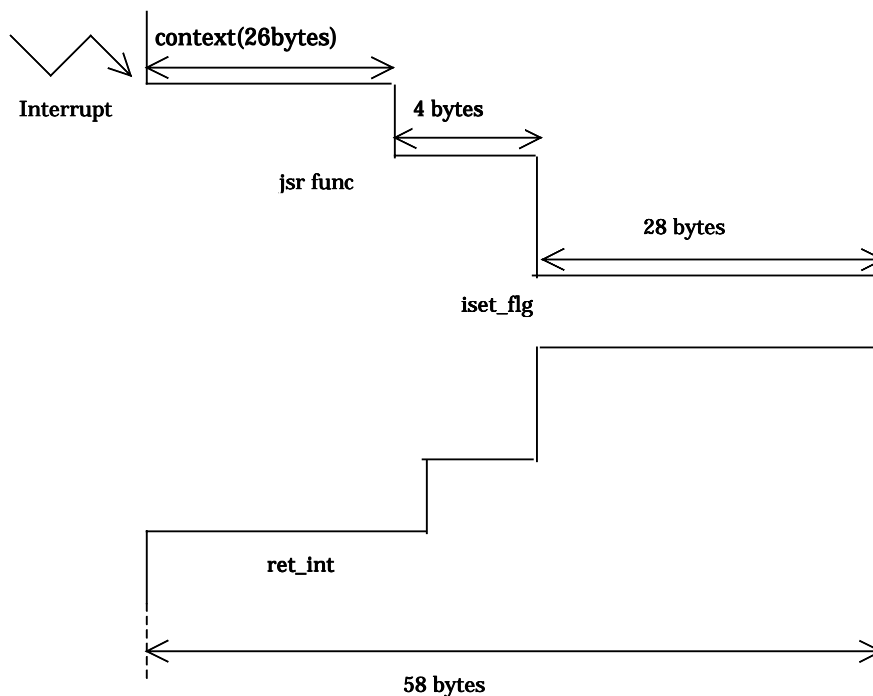

Figure 1.5: Stack size to be used by OS-dependent Interrupt Handler

[9] STK Viewer is a utility to calculate the stack size included with Renesas C Compiler NC308WA.

[(System stack size γ used by system clock interrupt handler)]

When you do not use a system timer, there is no need to add a system stack used by the system clock interrupt handler.

The system stack size γ used by the system clock interrupt handler is whichever larger of the two cases below:

- **38 + maximum size used by cyclic handler**
- **36 + maximum size used by alarm handler**

- **C language**
Using the stack size calculation utility STK Viewer[10], calculate the stack size of each Alarm or Cyclic handler.

- **Assembly language**
The stack size to be used by Alarm or Cyclic handler

= **register to be used** + **user size** + **stack size to be used by system call**

If neither cyclic handler nor alarm handler is used, then

$\gamma = 26$ bytes

When using the interrupt handler and system clock interrupt handler in combination, add the stack sizes used by both.

[10] STK Viewer is a utility to calculate the stack size included with Renesas C Compiler NC308WA.

Chapter 2 **System Call Reference**

2.1. Task Management Functions

2.1.1. sta_tsk(Start Task)

[(System call name)]

```
sta_tsk        →     Starts the task.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER sta_tsk (tskid, stacd);
```

<< Argument >>

```
ID      tskid;    The ID No. of the task to be started
INT     stacd;    Task start code
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
sta_tsk  tskid, stacd
```

<< Argument >>

```
tskid    [---*]     The ID No. of the task to be started
stacd    [--**]     Task start code
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	Task start code
A0	The ID No. of the task to be started

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_OBJ         0FFC1H(-H'003f):      Invalid object state
```

[(Function description)]

This system call starts the task indicated by tskid. That is, the specified task is put from the DORMANT state to the READY state or the RUN state. The startup code stacd is 16 bits.

In a C language program, stacd is passed to the startup task as an argument. In an assembly language program, stacd is stored in the startup task's R0 register.

This system call is valid only when the specified task is idle (DORMANT). Therefore, if a request is issued when the task is not idle (DORMANT), an error E_OBJ is returned to the system call issued task.

When this system call is issued, a task is activated by the start address and priority defined in the configuration file.[11]

If a task is reactivated after being terminated by ter_tsk or ext_tsk, it starts under the following conditions:[12]

- The task starts from the start address set in the configuration file.
- The wakeup request count is cleared to 0.
- The priority is the initial priority specified in the configuration file
- The initial register values except PC, R0(stacd), SB and FLG register are indeterminate.

Register	Initial register values
R0	Task start code
R1	indeterminate value
R2	indeterminate value
R3	indeterminate value
A0	indeterminate value
A1	indeterminate value
SB	__SB__
FB	indeterminate value
FLG	C0H
PC	Task start address

This system call can be issued from only tasks. If you want it to be issued from the interrupt handler, cyclic handler, or alarm handler, you must use a ista_tsk system call.

[11] However,the specified task does not always operate immediately.The operation always depends on the state of the ready queue at that time.
[12] Namely,the task totally starts from the reset state.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
   sta_tsk( ID_task2, stacd );
       :
}
void task2(int msg)
{
   if(msg == 0)
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task,task2
task:
   sta_tsk #ID_task2, msg
       :
task2:
   cmp.w       #0,R0
       :
```

2.1.2. ista_tsk(Start Task)

[(System call name)]

```
ista_tsk        →    Starts the task(for the handler only).
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ista_tsk (tskid, stacd);
```

<< Argument >>

```
ID       tskid;     The ID No. of the task to be started
INT      stacd;     Task start code
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
ista_tsk tskid, stacd
```

<< Argument >>

```
tskid    [---*]     The ID No. of the task to be started
stacd    [--**]     Task start code
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	Task start code
A0	The ID No. of the task to be started

[(Error codes)]

```
E_OK        00000H(-H'0000):      Normal End
E_OBJ       0FFC1H(-H'003f):      Invalid object state
```

[(Function description)]

Use this system call when you want to use the same function as that of the sta_tsk system call from the interrupt handler, cyclic handler, or alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
   ista_tsk( ID_task2, stacd );
        :
}
void task2(int msg)
{
   if(msg == 0)
        :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        intr
intr:
   ista_tsk #ID_task2, msg
        :
   ret_int
```

2.1.3. ext_tsk(Exit Task)

[(System call name)]

```
ext_tsk          →    Ends the own task.
```

[(Calling by the C language)]

```
#include <mr308.h>
void ext_tsk ();
```

<< Argument >>

None

<< Return value >>

Control is not returned to the task which issued this system call

[(Calling by the assembly language)]

```
.include mr308.inc
ext_tsk
```

<< Argument >>

None

<< Register setting >>

Control is not returned to the task which issued this system call

[(Error codes)]

Control is not returned to the task which issued this system call

[(Function description)]

This system call ends the own task; that is, it puts the own task from the RUN state to the DORMANT state. Once a task has been terminated, it does not operate until activated again by the sta_tsk or ista_tsk system call. When a task is activated again in this way, it can be started only from the start address defined in the configuration file.

That is, a task terminated by ext_tsk and then activated by sta_tsk operates as if it was reset. When this system call is issued, the semaphore obtained by the own task is not freed.

When issuing loc_cpu or dis_dsp, make sure you then issue unl_cpu or ena_dsp before ending the task (issue ext_tsk).

This system call can be issued only from tasks. It cannot be issued from the interrupt handler, the cyclic handler, and the alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
void task(void)
{
       :
   ext_tsk();
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   ext_tsk
```

2.1.4. ter_tsk(Terminate Task)

[(System call name)]

```
ter_tsk          →    Terminates a task forcibly.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ter_tsk (tskid);
```

<< Argument >>

```
ID      tskid;     The ID No. of the task to be forcibly terminated
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
ter_tsk  tskid
```

<< Argument >>

```
tskid    [---*]     The ID No. of the task to be forcibly terminated
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be forcibly terminated

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_OBJ         0FFC1H(-H'003f):      Invalid object state
```

[(Function description)]

The task indicated by tskid is forcibly terminated.

This system call cannot specify the own task. To terminate the own task, use the ext_tsk system call.

If a specified task is in WAIT state being linked to some waiting queue[13] the task is removed from the queue by execution of this system call. However, the semaphores, etc. that have been acquired by the specified task before that are not relinquished.

If the task indicated by tskid is in DORMANT state, the system returns an error E_OBJ for the system call.

This system call can be issued only from tasks. It cannot be issued from the interrupt handler, the cyclic handler, and the alarm handler.

[13] Timeout wait queue, eventflag wait queue, semaphore wait queue, or mail box wait queue is possible.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   ter_tsk( ID_main );
       :
   ext_tsk();
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   ter_tsk #ID_task2
       :
   .GLB        task2
task2:
       :
```

2.1.5. dis_dsp(Disable Dispatch)

[(System call name)]

```
dis_dsp        →    Disable dispatch of the task.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER dis_dsp ();
```

<< Argument >>

None

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
dis_dsp
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

Disables task dispatch.

After executing this system call, task dispatch is disabled until the ena_dsp system call is executed. Therefore, even when a task with higher priority than the task that executed dis_dsp by a system call issued from an interrupt handler or a task that executed dis_dsp is placed in READY state, no time is dispatched to that task. Namely, dispatching to tasks with higher priority is delayed until the dispatch disabled condition is terminated. However, since external interrupts are not disabled, an interrupt handler is activated even while dispatch is disabled.

If a task already in a dispatch disabled state issues dis_dsp, no error is assumed; the result is only that the dispatch disabled state continues. However, a dispatch disabled state is cleared by issuing only one ena_dsp no matter how many times dis_dsp may have been issued.

When issuing loc_cpu or dis_dsp, make sure you then issue unl_cpu or ena_dsp before ending the task (issue ext_tsk).

This system call can only be issued from tasks, and cannot be issued from the interrupt handler, cyclic handler, or alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   dis_dsp();
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   dis_dsp
       :
```

2.1.6. ena_dsp(Enable Dispatch)

[(System call name)]

ena_dsp → Permits dispatch of the task.

[(Calling by the C language)]

```
#include <mr308.h>
ER ena_dsp();
```

<< Argument >>

None

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
ena_dsp
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

Enables task dispatch.

Namely, it clears a dispatch disabled state set by dis_dsp, thereby activating the scheduler. If a task not in a dispatch disabled state issues ena_dsp, no error assumed; the result is only that the dispatch enabled state continues.

This system call can only be issued from tasks, and cannot be issued from the interrupt handler, cyclic handler, or alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   ena_dsp();
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   ena_dsp
       :
```

2.1.7. chg_pri(Change Task Priority)

[(System call name)]

chg_pri → Changes the priority of a task.

[(Calling by the C language)]

```
#include <mr308.h>
ER chg_pri (tskid,tskpri);
```

<< Argument >>

```
ID      tskid;    The ID No. of the task whose priority is changed
PRI     tskpri;   The priority to be changed
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
chg_pri  tskid, tskpri
```

<< Argument >>

```
tskid    [---*]    The ID No. of the task whose priority is changed
tskpri   [--**]    The priority to be changed
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The priority to be changed
A0	The ID No. of the task whose priority is changed (include TSK_SELF)

[(Error codes)]

```
E_OK       00000H(-H'0000):    Normal End
E_OBJ      0FFC1H(-H'003f):    Invalid object state
```

[(Function description)]

Changes the priority of the task indicated by tskid to a value indicated by tskpri. Furthermore, the task is rescheduled according to the result of this modification. If this system call is executed for a task linked to the ready queue (including a task in RUN state) or a task being queued in order of priorities, the task is moved to the tail of the queue of the relevant priority. Similarly, if the same priority as the previous one is specified, the task is moved to the tail of the queue of that priority.[14]

Task priority is higher when its number is lower. Priority 1 is the highest. The minimum value that can be specified for a priority is 1. The maximum value is the one specified in the configuration file. The range of the specifiable priority is 1 to 255.

For example, when the following is specified in the configuration file, the range of the specifiable priorities is 1 to 13 [15]

```
system{
        stack_size      = 0x100;
        priority        = 13;
};
```

If you specify tskid = TSK_SELF = 0, it specifies the task itself. This system call cannot be used to change the priority of a task in DORMANT state. There fore, if the task indicated by tskid is in DORMANT state, the system returns an error E_OBJ for the system call.

This system can be issued from only tasks. If you want it to be issued from the interrupt handler, cyclic handler, or alarm handler, you must use a ichg_pri system call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   chg_pri (ID_task2, 2);
       :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
       :
   chg_pri #ID_task2,#2
       :
```

[14] Therefore, by issuing this system call to set the same priority as the current one for the task itself, you can in effect relinquish control of execution of the task.

[15] Switchover to a task with lower priority calls for greater processing time and greater interrupt disabled time. Therefore, the narrower the priority range, the better. So reduce the priority range to a possible minimum.

2.1.8. ichg_pri(Change Task Priority)

[(System call name)]

ichg_pri → Changes the priority of a task (for the handler only).

[(Calling by the C language)]

```
#include <mr308.h>
ER ichg_pri (tskid,tskpri);
```

<< Argument >>

```
ID      tskid;     The ID No. of the task whose priority is changed
PRI     tskpri;    The priority to be changed
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
ichg_pri tskid, tskpri
```

<< Argument >>

```
tskid     [---*]     The ID No. of the task whose priority is changed
tskpri    [--**]     The priority to be changed
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The priority to be changed
A0	The ID No. of the task whose priority is changed

[(Error codes)]

```
E_OK      00000H(-H'0000):     Normal End
E_OBJ     0FFC1H(-H'003f):     Invalid object state
```

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the chg_pri system call.

In this system call, you cannot use tskid = TSK_SELF = 0 to specify the own task.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
        :
   ichg_pri(ID_main, 2);
        :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        intr
intr:
        :
   ichg_pri #ID_task2, #2
        :
   ret_int
```

2.1.9. rot_rdq(Rotate Ready Queue)

[(System call name)]

rot_rdq → Rotates the ready queue of a task.

[(Calling by the C language)]

```
#include <mr308.h>
ER rot_rdq (tskpri);
```

<< Argument >>

PRI tskpri; The priority of the ready queue to be rotated

<< Return value >>

E_OK is always returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
rot_rdq tskpri
```

<< Argument >>

tskpri [—**] The priority of the ready queue to be rotated

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The priority of the ready queue to be rotated (include TPRI_RUN)
A0	--

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

This system call rotates the ready queue having the priority specified by tskpri. That is, this system call reconnects the task linked to the head of the ready queue having the specified priority to the end of it in order to switch between the tasks having the same priority. See Figure 2.1.

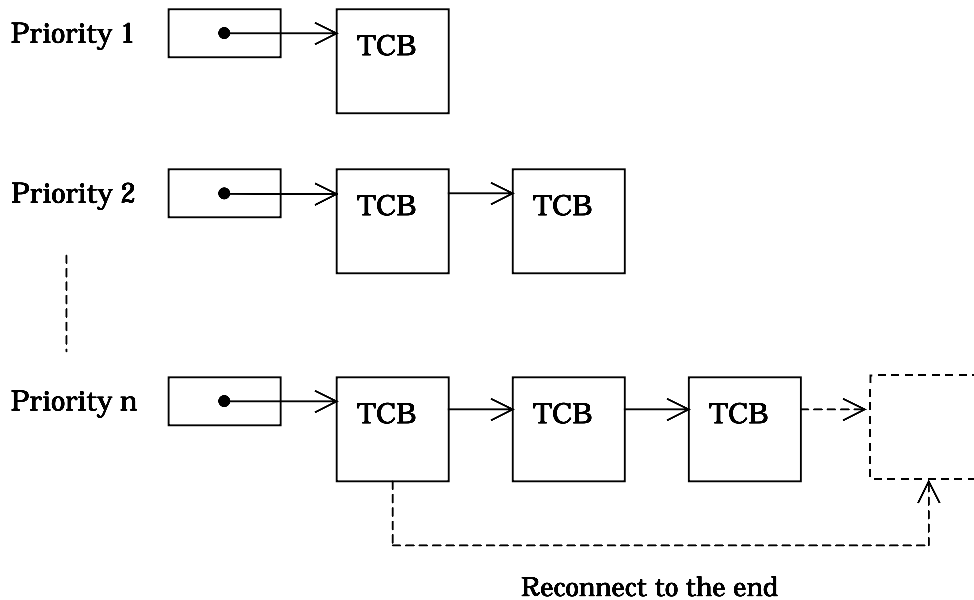

Figure 2.1: Ready Queue Operation by rot_rdq System Call

Issuing this system call at a certain interval allows round robin scheduling.

Specification tskpri = TPRI_RUN = 0 causes the ready queue with the priority of the own task to be rotated.

If this system call is used to specify the priority of the own task, the task is moved to the tail of that ready queue.

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the irot_rdq.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
void task()
{
        :
    rot_rdq(2);
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    rot_rdq #2
        :
```

2.1.10. irot_rdq(Rotate Ready Queue)

[(System call name)]

irot_rdq → Rotates the ready queue of a task (for the handler only).

[(Calling by the C language)]

```
#include <mr308.h>
ER irot_rdq (tskpri);
```

<< Argument >>

PRI tskpri; The priority of the ready queue to be rotated

<< Return value >>

E_OK is always returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
irot_rdq tskpri
```

<< Argument >>

tskpri [—**] The priority of the ready queue to be rotated

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The priority of the ready queue to be rotated (include TPRI_RUN)
A0	--

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the rot_rdq system call. If irot_rdq (tskpri = TPRI_RUN) is issued, the ready queue of the priority equal to that the task that was executing when the interrupt handler was invoked is rotated.

Issuing this system call allows round robin scheduling.

[(Usage example)]

In this example, round robin scheduling is implemented by rotating the ready queue having priority 2 at a certain intervals by the cyclic handler.

<< Usage example of the C language >>

```
#include <mr308.h>
void cyc()
{
        :
    irot_rdq(2);
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        cyc
cyc:
        :
    irot_rdq #2
        :
```

2.1.11. rel_wai(Release Task Wait)

[(System call name)]

```
rel_wai        →    Releases the task WAIT state forcibly.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER rel_wai (tskid);
```

<< Argument >>

```
ID       tskid;     The ID No. of the task to be forcibly released
                    from the WAIT state
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
rel_wai  tskid
```

<< Argument >>

```
tskid    [–––*]     The ID No. of the task to be forcibly released
                    from the WAIT state
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be forcibly released from the WAIT state

[(Error codes)]

```
E_OK         00000H(-H'0000):     Normal End
E_OBJ        0FFC1H(-H'003f):     Invalid object state
```

[(Function description)]

This system call unconditionally releases the task specified by tskid from the WAIT state(Except SUSPEND state). Error E_RLWAI is returned to the released task.

If the task is linked to some waiting queue, the task is removed from the queue [16] by execution of this system call. If the task is not in WAIT state, the system returns an error E_OBJ to the system call issued task.

This system call cannot specify the own task.

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the irel_wai.

[16] Timeout wait queue, eventflag wait queue, semaphore wait queue, or mail box wait queue is possible.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
            :
    if( rel_wai( ID_main ) != E_OK )
        error("Can't rel_wai main()\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    rel_wai #ID_main
        :
```

2.1.12. irel_wai(Release Task Wait)

[(System call name)]

irel_wai → Releases the task WAIT state forcibly (for the handler only).

[(Calling by the C language)]

```
#include <mr308.h>
ER irel_wai (tskid);
```

<< Argument >>

ID tskid; The ID No. of the task to be forcibly released from the WAIT state

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
irel_wai tskid
```

<< Argument >>

tskid [---*] The ID No. of the task to be forcibly released from the WAIT state

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be forcibly released from the WAIT state

[(Error codes)]

E_OK	00000H(-H'0000):	Normal End
E_OBJ	0FFC1H(-H'003f):	Invalid object state

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the rel_wai system call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
       :
   if( irel_wai( ID_main ) != E_OK )
       error("Can't irel_wai task(2)\n");
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        intr
intr:
       :
   irel_wai #ID_main
       :
   ret_int
```

2.1.13. get_tid(Get Self Task ID)

[(System call name)]

```
get_tid          →    Gets the ID of the self task
```

[(Calling by the C language)]

```
#include <mr308.h>
ER get_tid (p_tskid);
```

<< Argument >>

```
ID      *p_tskid;   The variable in which the task ID is stored.
```

<< Return value >>

```
The returned function value is always E_OK.
The ID No. of the own task is set in the area indicated by p_tskid.
```

[(Calling by the assembly language)]

```
.include mr308.inc
get_tid
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID of the self task

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

Gets the ID No. of the own task.

FALSE = 0 is returned if the system call is issued from the interrupt handler, cyclic handler, or alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
void task()
{
   ID tskid;
       :
   get_tid(&tskid);
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   get_tid
       :
```

2.1.14. ref_tsk(Refer Task Status)

[(System call name)]

```
ref_tsk            →    Reference Task Status
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_tsk (pk_rtsk,tskid);
```

<< Argument >>

```
ID       tskid;       The ID No. of the task to Reference Task
T_RTSK   *pk_rtsk;    Packet address to Reference Task
```

<< Return value >>

```
An error code is returned as the return value of a function.
The structure indicated by pk_rtsk returns the following data.
 typedef struct t_rtsk {
    VP      exinf;          /* Extended information */
    PRI     tskpri;         /* Current task priority level */
    UINT    tskstat;        /* Task status */
    UINT    tskwait;        /* Reason for wait */
    ID      wid;            /* Wait object ID */
 } T_RTSK;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_tsk  tskid, pk_rtsk
```

<< Argument >>

```
taskid    [---*]    The ID No. of the task to Reference Task
pk_rtsk   [-***]    Packet address to Reference Task
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to Reference Task (iinclude TSK_SELF)
A1	Packet address to Reference Task

The area indicated by pk_rtsk returns the following information.
Offset

```
+0    exinf      Extended information
+4    tskpri     Current task priority level
+6    tskstat    Task status
+8    tskwait    Reason for wait
+10   wid        Wait object ID
```

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

Refers to the status of the task indicated by tskid then returns the following task information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- tskpri

  Returns the task priority level in tskpri

- tskstat

  Returns a value corresponding to the status of the specified task in tskstat

TTS_RUN	(0001H)	RUN state
TTS_RDY	(0002H)	READY state
TTS_WAI	(0004H)	WAIT state
TTS_SUS	(0008H)	SUSPEND state
TTS_WAS	(000CH)	WAIT-SUSPEND state
TTS_DMT	(0010H)	DORMANT state

- tskwait

  If the target task is in the wait state, the cause of the wait is returned in tskwait. The following shows the values of the respective causes.

TTW_SLP	(0001H)	Waiting with slp_tsk or tslp_tsk
TTW_DLY	(0002H)	Waiting with dly_tsk
TTW_FLG	(0010H)	Waiting with wai_flg or twai_flg
TTW_SEM	(0020H)	Waiting with wai_sem or twai_sem
TTW_MBX	(0040H)	Waiting with rcv_msg or trcv_msg

- wid

  If the target task is in the wait state, its object ID No. is returned in wid.

A task may specify itself by specifying tskid = TSK_SELF = 0. Note, however, an interrupt handler cannot specify itself by specifying tskid = TSK_SELF.

If ref_tsk is issued by the interrupt handler targeting the interrupted task the RUN status (TTS_RUN) is returned in tskstat.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    T_RTSK rtsk;
        :
    ref_tsk( &rtsk, ID_main );
        :
}
```

<< Usage example of the assembly language >>

```
rtsk:  .blkb   10

    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    ref_tsk #ID_task2, #rtsk
        :
```

2.2. Synchronization Functions Attached to Task

2.2.1. sus_tsk(Suspend Task)

[(System call name)]

```
sus_tsk          →    Puts a task in the SUSPEND state.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER sus_tsk (tskid);
```

<< Argument >>

```
ID        tskid;      The ID No. of the task to be put in the SUSPEND
                      state
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
sus_tsk  tskid
```

<< Argument >>

```
tskid     [–––*]      The ID No. of the task to be put in the SUSPEND
                      state
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be put in the SUSPEND state

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_QOVR        0FFB7H(-H'0049):      Queuing or nest overflow
E_OBJ         0FFC1H(-H'003f):      Invalid object state
```

[(Function description)]

This system call discontinues the execution of the task specified by tskid and puts it in the SUSPEND state.

The SUSPEND state is cleared by issuing the rsm_tsk system call. When the task specified by tskid is in the DORMANT state, error E_OBJ is returned as the system call return value. This system call cannot specify the own task.

The SUSPEND request nesting by this system call is not performed. Therefore, when the task specified by tskid is in the SUSPEND state, error E_QOVR is returned as the system call return value

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the isus_tsk.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
    if( sus_tsk( ID_main ) != E_OK )
        printf("Can't suspend task main()\n");
       :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
       :
    sus_tsk #ID_task2
       :
```

2.2.2. isus_tsk(Suspend Task)

[(System call name)]

isus_tsk → Puts a task in the SUSPEND state (for the handler only)

[(Calling by the C language)]

```
#include <mr308.h>
ER isus_tsk (tskid);
```

<< Argument >>

ID tskid; The ID No. of the task to be put in the SUSPEND state

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
isus_tsk tskid
```

<< Argument >>

tskid [―――*] The ID No. of the task to be put in the SUSPEND state

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be put in the SUSPEND state

[(Error codes)]

```
E_OK      00000H(-H'0000):   Normal End
E_QOVR    0FFB7H(-H'0049):   Queuing or nest overflow
E_OBJ     0FFC1H(-H'003f):   Invalid object state
```

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the sus_tsk system call.

Since this is a system call from a handler, it allows you to specify any task ID. Therefore, this system call be used to suspend an interrupted task.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
        :
    if( isus_tsk( ID_main ) != E_OK )
        printf("Can't suspend main()\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        intr
intr:
        :
    isus_tsk #ID_main
        :
    ret_int
```

2.2.3. rsm_tsk(Resume Task)

[(System call name)]

```
rsm_tsk          →    Resumes the task in the SUSPEND state.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER rsm_tsk (tskid);
```

<< Argument >>

```
ID        tskid;     The ID No. of the task to be taken from the
                     SUSPEND state
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
rsm_tsk  tskid
```

<< Argument >>

```
tskid     [---*]     The ID No. of the task to be taken from the
                     SUSPEND state
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be taken from the SUSPEND state

[(Error codes)]

```
E_OK        00000H(-H'0000):      Normal End
E_OBJ       0FFC1H(-H'003f):      Invalid object state
```

[(Function description)]

If the task indicated by tskid has been suspended by sus_tsk system call, this system call clears its forced wait state and restarts execution of the task. In this case, the task is linked at the tail of the ready queue.

For the request issued when the task is not in forced waiting (SUSPEND) or the DORMANT state, error code E_OBJ is returned to the task which issued the system call. Since this system call is intended for tasks in forced waiting (SUSPEND) or double waiting (WAIT-SUSPEND) states, it cannot be used to specify the own task.

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the irsm_tsk.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
        :
    if( rsm_tsk( ID_main ) != E_OK )
        printf("Can't resume main()\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    rsm_tsk #ID_task2
        :
```

2.2.4. irsm_tsk(Resume Task)

[(System call name)]

```
irsm_tsk          →    Resumes the task in the SUSPEND state (for the
                       handler only).
```

[(Calling by the C language)]

```
#include <mr308.h>
ER irsm_tsk (tskid);
```

<< Argument >>

```
ID        tskid;     The ID No. of the task to be taken from the
                     SUSPEND state
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
irsm_tsk tskid
```

<< Argument >>

```
tskid     [---*]     The ID No. of the task to be taken from the
                     SUSPEND state
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be taken from the SUSPEND state

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
E_OBJ         0FFC1H(-H'003f):       Invalid object state
```

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the rsm_tsk system call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
       :
   irsm_tsk( ID_main );
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        intr
intr:
   :
irsm_tsk #ID_main
   :
```

2.2.5. slp_tsk(Sleep Task)

[(System call name)]

```
slp_tsk          →    Puts the task in the WAIT state.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER slp_tsk ();
```

<< Argument >>

None

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
slp_tsk
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
E_RLWAI       0FFAAH(-H'0056):       Wait state forcibly cleared
```

[(Function description)]

This system call puts the self task from the RUN state to the WAIT state. The WAIT state is cleared by the system call of the task wakeup issued for this task[17] or the system call which forcibly clears the WAIT state. [18] In the former, error code E_OK is returned; in the latter, error code E_RLWAI is returned.

When a task put in the WAIT state by slp_tsk is suspended (sus_tsk) by another task, that task is put in the WAIT-SUSPEND state. In this case, the task is still in the SUSPEND state even if the WAIT state is cleared by the system call of task wakeup and the execution of the task is not resumed until the rsm_tsk system call is issued.

This system call can be issued only from tasks.

[17] wup_tsk,iwup_tsk System call
[18] rel_wai,irel_wai System call

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   if( slp_tsk() != E_OK )
       error("Forced wakeup\n");
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE mr308.inc
   .GLB   task
task:
       :
   slp_tsk
       :
```

2.2.6. tslp_tsk(Sleep Task with Timeout)

[(System call name)]

```
tslp_tsk         →    Switches the task to the fixed-time wait state
```

[(Calling by the C language)]

```
#include <mr308.h>
ER tslp_tsk (tmout);
```

<< Argument >>

```
TMO       tmout;     Timeout value
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
tslp_tsk tmout
```

<< Argument >>

```
tmout     [--**]      Timeout value
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
R3	Timeout value

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
E_TMOUT       0FFABH(-H'0055):       Polling failed or timeout
E_RLWAI       0FFAAH(-H'0056):       Wait state forcibly cleared
```

[(Function description)]

Switches the task from the (RUN) status in which it runs for the specified time only to the WAIT state.

A wait state invoked by this system call is cancelled in the following cases:

- When a system call[19] to start a task is invoked from another task or interrupt.

  Error code E_OK is returned.

- When a system call[20] to forcibly cancel the wait state is invoked from another task or interrupt.

  Error code E_RLWAI is returned.

- When the time specified in tmout has elapsed. When the tmout is 0 and the value of the wakeup request count count is 0.

  Error code E_TMOUT is returned.

The unit of time specified in tmout is the unit of time of the system clock, specified in the configuration file.

tslp_tsk(10);

For example, if it is 10ms and the following is written in the program the own task is placed from the execution (RUN) state into a wait (WAIT) state and held in that state for 100 ms.

You can specify a timeout (tmout) of -1 to 0x7FFF. Specifying TMO_FEVR = -1 can be used to set the timeout period to forever (no timeout). In this case, tslp_tsk will function exactly the same as slp_tsk causing the issuing task to wait forever for wup_tsk to be issued.

This system call can be issued only from tasks.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
        :
    if( tslp_tsk( 10 ) != E_TMOUT )
        printf("Forced wakeup\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE mr308.inc
    .GLB task
task:
        :
    tslp_tsk #200
        :
```

[19] wup_tsk, iwup_tsk System call
[20] rel_wai, irel_wai System call

2.2.7. wup_tsk(Wakeup Task)

[(System call name)]

wup_tsk → Wakes up the task in the wait state.

[(Calling by the C language)]

```
#include <mr308.h>
ER wup_tsk (tskid);
```

<< Argument >>

ID tskid; The ID No. of the task to be waked up

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
wup_tsk  tskid
```

<< Argument >>

tskid [---*] The ID No. of the task to be waked up

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be waked up

[(Error codes)]

```
E_OK      00000H(-H'0000):   Normal End
E_QOVR    0FFB7H(-H'0049):   Queuing or nest overflow
E_OBJ     0FFC1H(-H'003f):   Invalid object state
```

[(Function description)]

If the task specified by tskid is in a wait (WAIT) state entered by execution of slp_tsk,tslp_tsk this system call clears the task's wait state to place it in an executable (READY) or execution (RUN) state. Also,

if the task specified by tskid is in a double-wait (WAIT-SUSPEND) state, the system call only clears the wait state and places the task in a forced wait (SUSPEND) state.

For a request issued when the task is in an idle (DORMANT) state, an error E_OBJ is returned to the system call issued task.

Note also that this system call cannot specify the own task.

If this system call is issued for tasks that are not in a wait (WAIT) state entered by execution of slp_tsk, tslp_tsk or a double-wait (WAIT-SUSPEND) state, wakeup requests are accumulated. More specifically, the wakeup request count in the TCB[21] of the task is incremented by 1[22]

The maximum value of the wakeup request count is 0x7FFF(32767). If a wakeup request is issued beyond 0x7FFF(32767), the count remains 0x7FFF(32767) and error code E_QOVR is returned to the task which issued this system call.

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the iwup_tsk.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
    if( wup_tsk( ID_main ) != E_OK )
        printf("Can't wakeup main()\n");
       :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
       :
    wup_tsk #ID_task2
```

[21] Task Control Block.

[22] This wakeup request count stores the counts of wakeup requests that have not been serviced because the intended task was not in a wait (WAIT) or a double-wait (WAIT-SUSPEND) state when the wup_tsk or iwup_tsk system call was issued to wake it up. If the task is being placed in a wait state by a slp_tsk system call when the wakeup request count is more than 1, the wakeup request count is decremented by 1. In this case, the task does not actually enter the wait (WAIT) state. Tasks can only be placed in a wait (WAIT) state by a slp_tsk system call when the wakeup request count is 0.

2.2.8. iwup_tsk(Wakeup Task)

[(System call name)]

```
iwup_tsk        →    Wakes up the task in the wait state (for the
                     handler only).
```

[(Calling by the C language)]

```
#include <mr308.h>
ER iwup_tsk (tskid);
```

<< Argument >>

```
ID      tskid;     The ID No. of the task to be waked up
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
iwup_tsk tskid
```

<< Argument >>

```
tskid   [---*]     The ID No. of the task to be waked up
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the task to be waked up

[(Error codes)]

```
E_OK        00000H(-H'0000):      Normal End
E_QOVR      0FFB7H(-H'0049):      Queuing or nest overflow
E_OBJ       0FFC1H(-H'003f):      Invalid object state
```

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the wup_tsk system call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
    if( iwup_tsk( ID_main ) != E_OK )
        printf("Can't wakeup main()\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        intr
intr:
        :
    iwup_tsk #ID_main
        :
    ret_int
```

2.2.9. can_wup(Cancel Wakeup Task)

[(System call name)]

```
can_wup          →    Cancels a task wakeup request.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER can_wup (p_wupcnt,tskid);
```

<< Argument >>

```
INT    *p_wupcnt;   The variable to store the count of canceled
                    wakeup
ID     tskid;       The ID No. of the task whose wakeup request is
                    to be canceled
```

<< Return value >>

```
An error code is returned as the return value of a function.
The count of canceled wakeup requests is set to variable wupcnt
```

[(Calling by the assembly language)]

```
.include mr308.inc
can_wup  tskid
```

<< Argument >>

```
tskid    [---*]     The ID No. of the task whose wakeup request is
                    to be canceled
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The variable to store the count of canceled wakeup
A0	The ID No. of the task whose wakeup request is to be canceled (include TSK_SELF)

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
E_OBJ         0FFC1H(-H'003f):       Invalid object state
```

[(Function description)]

This system call clears the wakeup request count for the task specified by tskid to zero. In other words, because the task to be waked up by the wup_tsk, or iwup_tsk system call before issuing the can_wup system call was not in the WAIT or WAIT-SUSPEND state, the can_wup system call clears all the accu- mulated wakeup requests. For the return value of this system call, the wakeup request count before being cleared to zero, namely the canceled wakeup re-quest count, is returned.

For the request issued when the task whose wakeup request is to be canceled is in the dor-mant state, error code E_OBJ is returned to the task which issued this system call.

When issued from only the task, this system call can tskid=TSK_SELF=0 as the own task.

The return parameter value(*p_wupcnt) is undefined in all cases other than E_OK.

This system call can be issued from either tasks or handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
void task()
{
    INT wupcnt;
        :
    if( can_wup(&wupcnt,ID_main) != E_OK )
        printf("Can't cancle wakeup main() \n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    can_wup #ID_task2
        :
```

2.3. Eventflags

2.3.1. set_flg(Set Eventflag)

[(System call name)]

```
set_flg          →    Sets an eventflag.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER set_flg (flgid, setptn);
```

<< Argument >>

```
ID      flgid;    The ID No. of the eventflag to be set
UINT    setptn;   The bit pattern to be set
```

<< Return value >>

```
E_OK is always returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
set_flg  flgid, setptn
```

<< Argument >>

```
flgid     [−−−*]     The ID No. of the eventflag to be set
setptn    [−−**]     The bit pattern to be set
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The bit pattern to be set
A0	The ID No. of the eventflag to be set

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

Among the 16-bit eventflags indicated by flgid, this system call sets the bit that is indicated by setptn. Namely, it logical OR's the value of the eventflags indicated by flgid with setptn.

If, as a result of the modification of the eventflag value, conditions to stop waiting are met for the task that has been kept waiting for that eventflag by a wai_flg system call, the task is freed from a wait state.

Multiple tasks can be kept waiting for the same eventflag. In this case, the multiple tasks can be simultaneously freed from a wait state by one issuance of a set_flg system call. However, if a task in a waiting queue was waiting for the eventflag to be set by a clear specification, all tasks up to that task are freed from the wait state.

If all bits in setptn are set to 0, no operation will be performed on the eventflag concerned; but this does not result in an error.

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the iset_flg system call.

[(Usage example)]

If the eventflag pattern before issuing this system call was 0xf, the pattern after this system

call becomes 0xff.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task(void)
{
       :
   set_flg( ID_flg,(UINT)0xf0 );
       :
   ext_tsk();
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   set_flg #ID_flg,#0f0H
       :
   ext_tsk
```

2.3.2. iset_flg(Set Eventflag)

[(System call name)]

```
iset_flg        →    Sets an eventflag (for the handler only).
```

[(Calling by the C language)]

```
#include <mr308.h>
ER iset_flg (flgid, setptn);
```

<< Argument >>

```
ID        flgid;     The ID No. of the eventflag to be set
UINT      setptn;    The bit pattern to be set
```

<< Return value >>

```
E_OK is always returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
iset_flg  flgid, setptn
```

<< Argument >>

```
flgid     [---*]     The ID No. of the eventflag to be set
setptn    [--**]     The bit pattern to be set
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The bit pattern to be set
A0	The ID No. of the eventflag to be set

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the set_flg system call.

[(Usage example)]

If the eventflag pattern before issuing this system call was 0xf, the pattern after this system call becomes 0xff.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand(void)
{
       :
   iset_flg( ID_flg, (UINT)0xf0);
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        intr
intr:
       :
   iset_flg  #ID_flg,#0f0H
       :
   ret_int
```

2.3.3. clr_flg(Clear Eventflag)

[(System call name)]

```
clr_flg          →    Clears an eventflag.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER clr_flg (flgid, clrptn);
```

<< Argument >>

```
ID       flgid;     The ID No. of the eventflag to be cleared
UINT     clrptn;    The bit pattern to be cleared
```

<< Return value >>

E_OK is always returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
clr_flg  flgid, clrptn
```

<< Argument >>

```
flgid     [---*]      The ID No. of the eventflag to be cleared
clrptn    [--**]      The bit pattern to be cleared
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The bit pattern to be cleared
A0	The ID No. of the eventflag to be cleared

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

Among the 16-bit eventflags indicated by flgid, this system call clears the bit whose corresponding clrptn is zero.

Namely, it logical AND's the value of the eventflags indicated by flgid with the value of clrptn. If all bits in clrptn are set to 1, no operation will be performed on the eventflag concerned; but this does not result in an error.

This system call can be issued from both tasks and handlers.

[(Usage example)]

If the eventflag pattern issuing this system call was 0xff, the pattern after this system call becomes 0xf0.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task(void)
{
       :
   clr_flg( ID_flg, (UINT)0xfff0 );
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   clr_flg #ID_flg,#0fff0H
       :
```

2.3.4. wai_flg(Wait Eventflag)

[(System call name)]

```
wai_flg          →    Waits for an eventflag.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER wai_flg (p_flgptn, flgid, waiptn, wfmode);
```

<< Argument >>

```
UINT   *p_flgptn;   Start address of area to which bit pattern is
                    returned when wait state is cleared
ID     flgid;       The ID No. of the eventflag to waited for
UINT   waiptn;      The bit pattern to be waited for
UINT   wfmode;      Wait mode
```

<< Return value >>

An error code is returned as the return value of a function.
The bit pattern when the wait cleared to the area specified by p_flgptn.

[(Calling by the assembly language)]

```
.include mr308.inc
wai_flg  flgid, waiptn, wfmode
```

<< Argument >>

```
flgid     [---*]    The ID No. of the eventflag to waited for
waiptn    [--**]    The bit pattern to be waited for
wfmode    [--**]    Wait mode
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	Wait mode
R2	The bit pattern when wait state is cleared
A0	The ID No. of the eventflag to waited for

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_RLWAI       0FFAAH(-H'0056):      Wait state forcibly cleared
```

[(Function description)]

In eventflags indicated by flgid, this system call waits until the bit specified by waiptn is set according to wait clear conditions indicated by wfmode.

Specify the wait bit pattern in waiptn. Note that you cannot specify 0 (zero) in waiptn. If you specify 0, this system call does not perform any processing and no value is returned. However, in the μITRON specifications, error E_PAR is returned, and compatibility with other realtime OS would therefore be compromised.

Following specifications are made with wfmode:

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

```
TWF_ANDW AND wait
TWF_ORW  OR wait
TWF_CLR  Clear specification
```

Namely, these specifications have the following effects:

wfmode(wait mode)	Effects
TWF_ANDW	Waits until all bits specified by waiptn are set. (AND wait)
TWF_ANDW+TWF_CLR	Clears the eventflag value to 0 (all bits will be cleared to 0) when AND wait clear conditions are met for the bit specified by waiptn and the task is freed from a wait state.
TWF_ORW	Waits until any bit specified by waiptn is set. (OR wait)
TWF_ORW+TWF_CLR	Clears the eventflag value to 0(all bits will be cleared to 0) when OR wait clear conditions are met for the bit specified by waiptn and the task is freed from a wait state.

flgptn is a return parameter that indicates the eventflag value before a wait state is cleared by this system call (in the case of a clear specification, the value of the eventflag before it is cleared). The value returned by flgptn is a value that satisfies wait clear conditions. Multiple tasks can be kept waiting for the same eventflag.

In this case, the multiple tasks can be simultaneously freed from a wait state by one issuance of a set_flg system call. However, if it was a task whose wait clear conditions are met in a waiting queue that requested a clear specification, all tasks up to that task are freed from the wait state.

The eventflag forms the queue of the tasks which perform the following operations:

- The order of queuing is FIFO (First In, First Out).
- If the queue has the task having clear specification, the flag is cleared when that task is cleared of the wait.
- Whether the tasks that follow the task having clear specification are cleared of wait or not depends on the eventflag already cleared. So, these tasks are not cleared of wait.

If the wait state is forcibly cleared by the rel_wai system call issued by another task, error code E_RLWAI is returned.

The return parameter value(*p_flgptn) is undefined in all cases other than E_OK.

This system call can be issued only from tasks.

[(Usage example)]

In this example, the system call waits until the bit specified by an eventflag whose flag name is flg2 is set. The task for which the specified bit is set is freed from a wait state.

Since the wait mode specified here is a clear specification, the eventflag flg2 is cleared to 0 simultaneously when the task is freed from a wait state.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    UINT flgptn;
        :
    if(wai_flg(&flgptn,ID_flg2,(UINT)0x0ff0,TWF_ANDW+TWF_CLR)!=E_OK)
        error("Wait Released\n");
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    wai_flg #ID_flg2,#0ff0H,#(TWF_ANDW+TWF_CLR)
        :
```

2.3.5. twai_flg(Wait Eventflag with Timeout)

[(System call name)]

twai_flg → Waits for an eventflag. (With Timeout)

[(Calling by the C language)]

```
#include <mr308.h>
ER twai_flg (p_flgptn, flgid, waiptn, wfmode, tmout);
```

<< Argument >>

```
UINT   *p_flptn;   Start address of area to which bit pattern is
                   returned when wait state is cleared
ID     flgid;      The ID No. of the eventflag to be waited for
UINT   waiptn;     The bit pattern to be waited for
UINT   wfmode;     Wait mode
TMO    tmout;      Timeout value
```

<< Return value >>

An error code is returned as the return value of a function.
The bit pattern when the wait cleared to the area specified by p_flgptn.

[(Calling by the assembly language)]

```
.include mr308.inc
twai_flg flgid, waiptn, wfmode, tmout
```

<< Argument >>

```
flgid    [---*]    The ID No. of the eventflag to be waited for
waiptn   [--**]    The bit pattern to be waited for
wfmode   [--**]    Wait mode
tmout    [--**]    Timeout value
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	Wait mode
R2	The bit pattern when wait state is cleared
R3	Timeout value
A0	The ID No. of the eventflag to be waited for

[(Error codes)]

```
E_OK        00000H(-H'0000):     Normal End
E_RLWAI     0FFAAH(-H'0056):     Wait state forcibly cleared
E_TMOUT     0FFABH(-H'0055):     Polling failed or timeout
```

[(Function description)]

In eventflags indicated by flgid, this system call waits until the bit specified by waiptn is set according to wait clear condition indicated by wfmode.

The task that invoked this system call is queued in two wait queues: the eventflag wait queue and timeout wait queue.

When this system call is invoked, the wait state is cancelled in the cases shown below. When the wait state is cancelled, the task that invoked this system call exits from the two wait queues (eventflag wait queue and timeout wait queue) and is connected to the ready queue.

- When the wait cancellation condition occurs before the tmout time has elapsed.

  Error code E_OK is returned.

● When the tmout time elapses without the wait cancellation condition being satisfied

Error code E_TMOUT is returned.

● When the wait state is forcibly cancelled by rel_wai or irel_wai system caLls being invoked from another task or handler.

Error code E_RLWAI is returned.

You can specify a timeout (tmout) of -1 to 0x7FFF. Specifying TMO_FEVR = -1 to twai_flg for tmout indicates that an infinite timeout value be used, resulting in exactly the same processing as wai_flg. If you specify tmout as TMO_POL(=0), it works like pol_flg.

See wai_flg system call for details of wfmode.

The return parameter value(*p_flgptn) is undefined in all cases other than E_OK.

This system call can be issued only from tasks. It cannot be issued from the interrupt handler, the cyclic handler, or the alarm handler.

[(Usage example)]

In this example, that task waits for the bit specified in the flg2 eventflag to be set or wait time tmout to elapse. The wait state is cancelled when the specified bit is set or the wait time has elapsed.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    UINT flgptn;
        :
    if( twai_flg(&flgptn, ID_flg2,(UINT)0x0ff0, TWF_ANDW, 5) != E_OK )
        error("Wait Released\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    twai_flg #ID_flg2,#0ff0H,#(TWF_ANDW+TWF_CLR),#5
        :
```

2.3.6. pol_flg(Poll Eventflag)

[(System call name)]

```
pol_flg          →    Gets an eventflag . (no wait state).
```

[(Calling by the C language)]

```
#include <mr308.h>
ER pol_flg (p_flgptn, flgid, waiptn, wfmode);
```

<< Argument >>

```
UINT   *p_flgptn;   Start address of area to which bit pattern is
                    returned when wait state is cleared
ID     flgid;       The ID No. of the eventflag to check
UINT   waiptn;      Wait bit pattern
UINT   wfmode;      Wait mode
```

<< Return value >>

Error code is returned as a return value for a numeral.
The bit pattern when a wait state is cleared is set in an area indicated by p_flgptn.

[(Calling by the assembly language)]

```
.include mr308.inc
pol_flg  flgid, waiptn, wfmode
```

<< Argument >>

```
flgid     [---*]     The ID No. of the eventflag to check
waiptn    [--**]     Wait bit pattern
wfmode    [--**]     Wait mode
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	Wait mode
R2	The bit pattern when wait state is cleared
A0	The ID No. of the eventflag to check

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_TMOUT       0FFABH(-H'0055):      Polling failed or timeout
```

[(Function description)]

In eventflags indicated by flgid, this system call checks to see if the wait clear bit pattern indicated by waiptn is set according to wfmode.

If the eventflag concerned already satisfies the wait clear conditions indicated by wfmode, the system call performs the same processing as in wai_flg (by clearing the eventflag if a clear specification is requested) and terminates the session normally.

If the eventflag concerned does not satisfy the wait clear conditions indicated by wfmode, the system call returns an error E_TMOUT. In this case, the task is not placed in a wait state. Nor is the eventflag cleared even if a clear specification is requested.

The return parameter value(*p_flgptn) is undefined in all cases other than E_OK.

This system call can be issued from both tasks and handlers.

[(Usage example)]

In this example, the system call examines whether the bit specified by an eventflag whose flag name is flg2 is set. Since a clear specification is requested, the eventflag is cleared to 0 if conditions are met.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    UINT flgptn;
        :
    if(pol_flg(&flgptn,ID_flg2,(UINT)0x0ff0,TWF_ORW+TWF_CLR)!=E_OK)
        printf("Not set EventFlag\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
    pol_flg #ID_flg2,#0ff0H,#(TWF_ORW+TWF_CLR)
        :
```

2.3.7. ref_flg(Refer Eventflag Status)

[(System call name)]

```
ref_flg          →    Reference Eventflag Status.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_flg (pk_rflg, flgid);
```

<< Argument >>

```
T_RFLG  *pk_rflg;   Packet address to Reference Eventflag
ID      flgid;      The ID No. of the eventflag to Reference
                    Eventflag
```

<< Return value >>

An error code is returned as the return value of a function.
The structure indicated by pk_rflg returns the following data.

```
typedef struct t_rflg {
    VP      exinf;  /* Extended information */
    BOOL_ID wtsk;   /* Waiting task information */
    UINT    flgptn; /* Bit pattern of Eventflag */
} T_RFLG;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_flg  flgid, pk_rflg
```

<< Argument >>

```
flgid     [---*]    The ID No. of the eventflag to Reference
                    Eventflag
pk_rflg   [-***]    Packet address to Reference Eventflag
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the eventflag to Reference Eventflag
A1	Packet address to Reference Eventflag

The area indicated by pk_rflg returns the following information.

offset

```
+0   exinf     Extended information
+4   wtsk      Waiting task information
+6   flgptn    Bit pattern of Eventflag
```

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
```

[(Function description)]

Refers to the state of the eventflag specified by flgid, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- wtskid

  wtsk returns the ID No. of the first task (the first task to enter the wait state) in the wait queue. wtsk returns FALSE(0) if there are no tasks waiting in the queue.

- flgptn

  flgptn returns the current value of the eventflag.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    T_RFLG rflg;
    ref_flg(&rflg, ID_flg );
        :
}
```

<< Usage example of the assembly language >>

```
 rflg:  .blkb   8

    .INCLUDE    mr308.inc
    .GLB        task
 task:
        :
    ref_flg     #ID_flg, #rflg
        :
```

2.4. Semaphore

2.4.1. sig_sem(Signal Semaphore)

[(System call name)]

```
sig_sem          →    Returns resource to the semaphore
```

[(Calling by the C language)]

```
#include <mr308.h>
ER sig_sem (semid);
```

<< Argument >>

```
ID       semid;     The ID No. of the semaphore
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
sig_sem  semid
```

<< Argument >>

```
semid    [–––*]      The ID No. of the semaphore
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the semaphore

[(Error codes)]

```
E_OK           00000H(-H'0000):        Normal End
E_QOVR         0FFB7H(-H'0049):        Queuing or nest overflow
```

[(Function description)]

This system call returns 1 resource to the semaphore specified by semid.

When tasks are linked to the queue of that semaphore, the task at the head of the queue is put in the ready state. If no task is linked, the count of that semaphore is incremented by 1.[23]

The maximum count value of a semaphore is 0x7FFF(32767). If it returns resource (sig_sem or isig_sem system call) is executed beyond 0x7FFF(32767), error code E_QOVR is returned to the task which issued the system call with the semaphore count value left unchanged, namely 0x7FFF(32767).

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the isig_sem.

[23] If this system call causes the count value to exceeds the semaphore initial value defined in the configuration file, no error will occur.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
        :
    if( sig_sem( ID_sem ) != E_OK )
        error("Overflow\n”);
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
    sig_sem     #ID_sem
        :
```

2.4.2. isig_sem(Signal Semaphore)

[(System call name)]

isig_sem → Returns resource to the semaphore (For the handler only)

[(Calling by the C language)]

```
#include <mr308.h>
ER isig_sem (semid);
```

<< Argument >>

ID semid; The ID No. of the semaphore

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
isig_sem semid
```

<< Argument >>

semid [---*] The ID No. of the semaphore

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the semaphore

[(Error codes)]

E_OK 00000H(-H'0000): Normal End
E_QOVR 0FFB7H(-H'0049): Queuing or nest overflow

[(Function description)]

This system call is issued from the interrupt handler, the cyclic handler, or the alarm handler to provide the same functions as the sig_sem system call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void inthand()
{
        :
    if( isig_sem( ID_sem ) != E_OK )
        error("Overflow\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        intr
intr:
    isig_sem    #ID_sem
        :
    ret_int
```

2.4.3. wai_sem(Wait on Semaphore)

[(System call name)]

```
wai_sem          →    Obtains one resource from the semaphore.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER wai_sem (semid);
```

<< Argument >>

```
ID       semid;     The ID No. of the semaphore from which the
                    resource is obtained
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
wai_sem  semid
```

<< Argument >>

```
semid    [---*]     The ID No. of the semaphore from which the
                    resource is obtained
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the semaphore from which the resource is obtained

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_RLWAI       0FFAAH(-H'0056):      Wait state forcibly cleared
```

[(Function description)]

This system call obtains 1 resource from the semaphore specified by semid.

If the count value of that semaphore is one or more, the count is decremented by 1 and the task which issued the system call continues executing. Conversely, if the semaphore count value is 0, the count value is not modified and the system call issued task is linked to the semaphore queue in order of FIFO.[24]

If the wait state has been cleared by the irel_wai or rel_wai system call issued by another task, error code E_RLWAI is returned.

This system call can be issued only from tasks. It cannot be issued from the interrupt handler, the cyclic handler, or the alarm handler.

[24] First-in, first-out. Namely, tasks are freed from a wait state by sig_sem or isig_sem system calls in the order they were placed in a wait state by the wai_sem system call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
        :
    if( wai_sem( ID_sem ) != E_OK )
        printf("Forced wakeup\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
    wai_sem     #ID_sem
        :
```

2.4.4. twai_sem(Wait on Semaphore with Timeout)

[(System call name)]

```
twai_sem          →    Obtains one resource from the semaphore.  (With
                       Timeout)
```

[(Calling by the C language)]

```
#include <mr308.h>
ER twai_sem (semid,tmout);
```

<< Argument >>

```
ID        semid;     The ID No. of the semaphore from which the
                     resource
TMO       tmout;     Timeout value
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
twai_sem semid, tmout
```

<< Argument >>

```
semid    [−−−*]     The ID No. of the semaphore from which the
                    resource
tmout    [−−**]     Timeout value
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
R3	Timeout value
A0	The ID No. of the semaphore from which the resource

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_TMOUT       0FFABH(-H'0055):      Polling failed or timeout
E_RLWAI       0FFAAH(-H'0056):      Wait state forcibly cleared
```

[(Function description)]

This system call obtains 1 resource from the semaphore specified by semid.

If the count value of that semaphore is one or more, the count is decremented by 1 and the task which issued the system call continues executing.

Conversely, if the semaphore count value is 0, the count value is not modified and the system call issued task is linked to the semaphore queue and timeout wait queue.

When this system call is invoked, the wait state is cancelled in the cases shown below. When the wait state is cancelled, the task that invoked this system call exits from the two wait queues (semaphore queue and timeout wait queue) and is connected to the ready queue.

- When the wait cancellation condition occurs by sig_sem system call being invoked before the tmout time has elapsed.

  Error code E_OK is returned.

- When the tmout time elapses without the wait cancellation condition being satisfied

  Error code E_TMOUT is returned.

- When the wait state is forcibly cancelled by rel_wai or irel_wai system calls being invoked from another task or handler.

  Error code E_RLWAI is returned.

You can specify a timeout (tmout) of -1 to 0x7FFF. Specifying TMO_FEVR = -1 to twai_sem for tmout indicates that an infinite timeout value be used, resulting in exactly the same processing as wai_sem. If you specify tmout as TMO_POL(=0), it works like preq_sem.

This system call can be issued only from tasks. It cannot be issued from the in terrupt handler, the cyclic handler, or the alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
        :
    if( twai_sem( ID_sem, 10 ) != E_OK )
        printf("Forced wakeup\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    twai_sem    #ID_sem,#10
        :
```

2.4.5. preq_sem(Poll and Request Semaphore)

[(System call name)]

preq_sem → Obtains one resource from the semaphore. (no wait)

[(Calling by the C language)]

```
#include <mr308.h>
ER preq_sem (semid);
```

<< Argument >>

ID semid; The ID No. of the semaphore from which the resource is obtained

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
preq_sem semid
```

<< Argument >>

semid [−−−*] The ID No. of the semaphore from which the resource is obtained

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the semaphore from which the resource is obtained

[(Error codes)]

E_OK 00000H(-H'0000): Normal End
E_TMOUT 0FFABH(-H'0055): Polling failed or timeout

[(Function description)]

Obtains 1 resource (without a wait state) from the semaphore indicated by semid.

If the count value of the semaphore concerned is 1 or more, the count value is decremented by 1 and the system call issued task continues executing.

Conversely, if the semaphore count value is 0, the count value is not modified and an error E_TMOUT is returned to the system call issued task.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
    if( preq_sem( ID_sem ) != E_OK )
        printf("No more resource\n");
       :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
       :
    preq_sem    #ID_sem
       :
```

2.4.6. ref_sem(Refer Semaphore Status)

[(System call name)]

```
ref_sem          →    Reference Semaphore Status
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_sem(pk_rsem,semid);
```

<< Argument >>

```
T_RSEM  *pk_rsem;   Packet address to Reference Semaphore
ID      semid;      The ID No. of the semaphore to Reference
                    Semaphore
```

<< Return value >>

```
An error code is returned as the return value of a function.
The structure indicated by pk_rsem returns the following data.
 typedef struct t_rsem {
    VP       exinf;  /* Extended information */
    BOOL_ID wtsk;   /* Waiting task information */
    INT      semcnt; /* Current semaphore count */
 } T_RSEM;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_sem  semid, pk_rsem
```

<< Argument >>

```
semid     [---*]    The ID No. of the semaphore to Reference
                    Semaphore
pk_rsem   [-***]    Packet address to Reference Semaphore
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the semaphore to Reference Semaphore
A1	Packet address to Reference Semaphore

The area indicated by pk_rsem returns the following information.

offset

```
+0   exinf     Extended information
+4   wtsk      Waiting task information
+6   semcnt    Current semaphore count
```

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
```

[(Function description)]

Refers to the state of the semaphore specified by semid, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- wtsk

  wtsk returns the ID No. of the first task (the first task to enter the wait state) in the wait queue. wtsk returns FALSE(0) if there are no tasks waiting in the queue.

- semcnt

  semcnt returns the current semaphore count.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    T_RSEM   rsem;
         :
    ref_sem( &rsem, ID_sem );
         :
}
```

<< Usage example of the assembly language >>

```
rsem:  .blkb   8
    .INCLUDE    mr308.inc
    . GLB       task
task:
         :
    ref_sem     #ID_sem1,#rsem
         :
```

2.5. Mailbox

2.5.1. snd_msg(Send Message to Mailbox)

[(System call name)]

```
snd_msg          →    Sends a message.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER snd_msg (mbxid, pk_msg);
```

<< Argument >>

```
ID         mbxid;      The ID No. of the mailbox to which a message is
                       sent
T_MSG      *pk_msg;    The start address of message packet
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
snd_msg  mbxid, pk_msg
```

<< Argument >>

```
mbxid      [−−−*]      The ID No. of the mailbox to which a message is
                       sent
pk_msg     [−−**]      The start address of message packet (16-bit)
           [****]      The start address of message packet (32-bit)
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of message packet (lower)(only 32-bit)
R2	The start address of message packet (only 16-bit)
R3	The start address of message packet (upper)(only 32-bit)
A0	The ID No. of the mailbox to which a message is sent

[(Error codes)]

```
E_OK     00000H(-H'0000):        Normal End
E_QOVR   0FFB7H(-H'0049):        Queuing or nest overflow
```

[(Function description)]

This system call sends a message to the mailbox specified by mbxid.

If there are no tasks waiting for a message, the message is stored in the message queue in order of FIFO.[25] Therefore, messages are taken out of the queue in the order they were sent to the mail box by issuing this system call. If there is any task waiting for a message, the message is passed to that task and the task has its wait state removed.

The size of the message queue is defined in the configuration file.

If this system call is issued for a mail box whose message queue is full, an error E_QOVR is returned to the system call issued task.

A message is 16 bits or 32bits wide data.[26]In standard μITRON specifi- cations, this data is interpreted as indicating the start address of a message packet (a structure including the message), i.e., address transfer. In MR308, however, messages can be handled in two ways to perform data communication as described below.

1. Using a message as the start address (16 bits or 32 bits) of a message packet Since no specific types of message packets (T_MSG) are stipulated in MR308, any desired message type can be defined by the user. It can be an array, for example. [27]
   Example:

   ```
   typedef char T_MSG;
   ```

   Define the start address pk_msg of the message packet as follows:

   For a 16-bit data item

   ```
   T_MSG near * pk_msg;
   ```

   For a 32-bit data item

   ```
   T_MSG far * pk_msg;
   ```

2. Using a message simply as data
   In this case, cast the second argument of the snd_msg and isnd_msg system calls (message data pk_msg to be sent) with (PT_MSG) and the first argument of rcv_msg and prcv_msg (address ppk_msg of the area in which to store the message data) with (PT_MSG *), respectively.

   To send variable I of int type, for example, write your statement as follows:

   ```
   int i, j;
   snd_msg( ID_mbx, (PT_MSG)i );
   rcv_msg( (PT_MSG *)&j, ID_mbx );
   ```

   This allows you to send 16-bit data directly.

This system call can be issued only from tasks. The system call which be issued from the interrupt handler, the cyclic handler, or the alarm handler is the isnd_msg.

[25] First In First Out
[26] You choose which to use - 16-bit data width or 32-bit data width - in the configuration file.
[27] It is standard to send the start address of message packet in [Calling by the C language] of this manual.

[(Usage example)]

In this example, the message is used to send the start address of a message packet.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
typedef char T_MSG;
T_MSG far *msg;
void task(void)
{
        :
    if( snd_msg( ID_msg, msg ) != E_OK ){
        error("overflow\n");
    }
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
 msg:   .BLKL   1

 task:
    snd_msg     #ID_msg, msg
        :
```

2.5.2. isnd_msg(Send Message to Mailbox)

[(System call name)]

```
isnd_msg          →    Sends a message.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER isnd_msg (mbxid, pk_msg);
```

<< Argument >>

```
ID        mbxid;     The ID No. of the mailbox to which a message is
                     sent
T_MSG     *pk_msg;   The start address of message packet
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
isnd_msg mbxid, pk_msg
```

<< Argument >>

```
mbxid     [――*]      The ID No. of the mailbox to which a message is
                     sent
pk_msg    [―**]      The start address of message packet (16-bit)
          [****]     The start address of message packet (32-bit)
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of message packet (lower)(only 32-bit)
R2	The start address of message packet (only 16-bit)
R3	The start address of message packet (upper)(only 32-bit)
A0	The ID No. of the mailbox to which a message is sent

[(Error codes)]

```
E_OK        00000H(-H'0000):      Normal End
E_QOVR      0FFB7H(-H'0049):      Queuing or nest overflow
```

[(Function description)]

This system call is used when using the function of the snd_msg system call from an task-independent section (e.g., interrupt handler, cyclic handler, or alarm handler).

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
typedef char T_MSG;
T_MSG msg[10];
void inthand()
{
       :
   if( isnd_msg(ID_msg, msg) != E_OK ) {
       error("overflow\n");
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        intr
intr:
       :
   isnd_msg    #ID_msg, #1234H
       :
   ret_int
```

2.5.3. rcv_msg(Receive Message from Mailbox)

[(System call name)]

```
rcv_msg          →    Waits for receiving a message.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER rcv_msg (ppk_msg, mbxid);
```

<< Argument >>

```
ID       mbxid;       The ID No. of the mailbox from which a message
                      is received
T_MSG    **ppk_msg;   The pointer variable to indicate the start
                      address of message packet
```

<< Return value >>

An error code is returned as the return value of a function.
The start address of the received message packet is set to variable ppk_msg.

[(Calling by the assembly language)]

```
.include mr308.inc
rcv_msg  mbxid
```

<< Argument >>

```
mbxid    [---*]    The ID No. of the mailbox from which a message
                   is received
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of message packet (upper)(only 32-bit)
R2	The start address of message packet (lower)
A0	The ID No. of the mailbox to which a message is received

[(Error codes)]

```
E_OK        00000H(-H'0000):     Normal End
E_RLWAI     0FFAAH(-H'0056):     Wait state forcibly cleared
```

[(Function description)]

This system call receives a message from the mailbox specified by mbxid.

If messages have arrived at the mail box concerned, this system call gets 1 message from the top of the message queue and returns it as a return parameter pk_msg.

Conversely, if no message has reached the mail box, the task that has issued this system call is placed in a wait state and linked in a waiting queue in order of FIFO.

If the task is freed from a wait state by a rel_wai system call issued by some other task, an error E_RLWAI is returned.

The return parameter value(*ppk_msg) is undefined in all cases other than E_OK.

This system call can only be issued from tasks.

Following precautions should observed when receiving a message:

1. When using a message as the start address of a message packet
   Since the message is 16 bits wide, declare the pointer variable (ppk_msg) to the area where the start address of a message packet is stored as follows:

   For a 16-bit data item

   ```
   T_MSG near * ppk_msg;
   ```

   For a 32-bit data item

   ```
   T_MSG far * ppk_msg;
   ```

   ```
   rcv_msg(&ppk_msg,ID_mbx);
   ```

2. When using a message simply as data
   Also, cast the first argument of rcv_msg and prcv_msg (address ppk_msg of the area in which to store the message data) with (PT_MSG *). id

   To send variable I of long type, for example:

   ```
   long i, j;
   snd_msg( ID_mbx, (PT_MSG)i );
   rcv_msg( (PT_MSG *)&j, ID_mbx );
   ```

[(Usage example)]

In this example, the message is used to send the start address of a message packet.

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
typedef T_MSG char;
void task()
{
    T_MSG *msg;
        :
    if( rcv_msg( &msg, ID_mbx ) != E_OK )
        error("forced wakeup\n");
        :
}
```

```
<< Usage example of the assembly language >>
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    rcv_msg     #ID_mbx
        :
```

2.5.4. trcv_msg(Receive Message with Timeout)

[(System call name)]

```
trcv_msg        →   Waits for receiving a message. (With Timeout)
```

[(Calling by the C language)]

```
#include <mr308.h>
ER trcv_msg (ppk_msg, mbxid, tmout);
```

<< Argument >>

```
ID      mbxid;      The ID No. of the mailbox from which a message
                    is received
T_MSG   **ppk_msg;  The pointer variable to indicate the start
                    address of message packet
TMO     tmout       Timeout value
```

<< Return value >>

An error code is returned as the return value of a function.
The start address of the received message packet is set to variable ppk_msg.

[(Calling by the assembly language)]

```
.include mr308.inc
trcv_msg mbxid,tmout
```

<< Argument >>

```
mbxid   [---*]    The ID No. of the mailbox from which a message
                  is received
tmout   [--**]    Timeout value
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of message packet (upper)(only 32-bit)
R2	The start address of message packet (lower)
R3	--
A0	The ID No. of the mailbox from which a message is re-ceived

[(Error codes)]

```
E_OK        00000H(-H'0000):    Normal End
E_TMOUT     0FFABH(-H'0055):    Polling failed or timeout
E_RLWAI     0FFAAH(-H'0056):    Wait state forcibly cleared
```

[(Function description)]

This system call receives a message from the mailbox specified by mbxid. If messages have arrived at the mail box concerned, this system call gets 1 mess age from the top of the message queue and returns it as a return parameter ppk_msg.

Conversely, if no message has reached the mail box, the task that has issued this system call is placed in a wait state and linked in a waiting queue and timeout wait queue.

When this system call is invoked, the wait state is cancelled in the cases shown below. When the wait state is cancelled, the task that invoked this system call exits from the two wait queues (message queue and timeout wait queue) and is connected to the ready queue.

- When the wait cancellation condition occurs by a message being received before the tmout time has elapsed.

  Error code E_OK is returned.

- When tmout time has elapsed without any message being received

  Error code E_TMOUT is returned.

- When the wait state is forcibly cancelled by rel_wai or irel_wai system calls being invoked from another task or handler.

  Error code E_RLWAI is returned.

You can specify a timeout (tmout) of -1 to 0x7FFF. Specifying TMO_FEVR = -1 to trcv_msg for tmout indicates that an infinite timeout value be used, resulting in exactly the same processing as rcv_msg. If you specify tmout as TMO_POL(=0), it works like prcv_msg.

See rcv_msg system call for precautions should observed when receiving a message.

The return parameter value(*ppk_msg) is undefined in all cases other than E_OK.

This system call can be issued only from tasks. It cannot be issued from the in terrupt handler, the cyclic handler, or the alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
typedef char T_MSG;
void task()
{
    T_MSG   *msg;
        :
    if( trcv_msg( &msg, ID_mbx, 10 ) != E_OK ){
        error("Can't Get Message\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
        :
    trcv_msg    #ID_mbx,#10
        :
```

2.5.5. prcv_msg(Poll and Receive Message)

[(System call name)]

```
prcv_msg        →    Receiving a message. (no wait)
```

[(Calling by the C language)]

```
#include <mr308.h>
ER prcv_msg (ppk_msg, mbxid);
```

<< Argument >>

```
ID      mbxid;       The ID No. of the mailbox from which a message
                     is received
T_MSG   **ppk_msg;   The start address of message packet
```

<< Return value >>

```
An error code is returned as the return value of a function.
The start address of the received message packet is set to variable
ppk_msg.
```

[(Calling by the assembly language)]

```
.include mr308.inc
prcv_msg mbxid
```

<< Argument >>

```
mbxid    [---*]    The ID No. of the mailbox from which a message
                   is received
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of message packet (upper)(only 32-bit)
R2	The start address of message packet (lower)
A0	The ID No. of the mailbox from which a message is received

[(Error codes)]

```
E_OK         00000H(-H'0000):     Normal End
E_TMOUT      0FFABH(-H'0055):     Polling failed or timeout
```

[(Function description)]

If any message is found in the mail box indicated by mbxid, this system call receives it (without a wait state). If the mail box contains messages, the system call gets 1 message from the top of the message queue and returns it as a return parameter ppk_msg.

Conversely, if no message has been sent to the mail box, an error E_TMOUT is returned to the system call issued task.

This system call can be issued from both a task and a task-independent section (e.g., interrupt handler, cyclic handler, or alarm handler).

The return parameter value(*ppk_msg) is undefined in all cases other than E_OK.

Refer to rcv_msg for precautions to be observed when receiving a message.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
typedef char T_MSG;
void task()
{
    T_MSG * msg;
        :
    if( prcv_msg( &msg, ID_mbx ) != E_OK ){
        error("Can't Get Message\n”);
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
    prcv_msg    #ID_mbx1
        :
```

2.5.6. ref_mbx(Refer Mailbox Status)

[(System call name)]

```
ref_mbx          →    Reference Mailbox Status
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_mbx (pk_rmbx, mbxid);
```

<< Argument >>

```
T_RMBX   *rmbx;    Packet address to Reference Mailbox
ID       mbxid;    The ID No. of the mailbox to Reference Maibox
```

<< Return value >>

An error code is returned as the return value of a function.
The structure indicated by pk_rmbx returns the following data.

```
typedef struct t_rmbx {
    VP       exinf;  /* Extended informatio */
    BOOL_ID  wtsk;   /* Waiting task information */
    T_MSG    *pk_msg; /* Starting address of next received message
                        packet*/
    INT      msgcnt; /* The number of messages */
} T_RMBX;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_mbx  mbxid, pk_rmbx
```

<< Argument >>

```
mbxid     [---*]    The ID No. of the mailbox to Reference Maibox
pk_rmbx   [-***]    Packet address to Reference Mailbox
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the mailbox to Reference Maibox
A1	Packet address to Reference Mailbox

The structure indicated by pk_rmbx returns the following data.

offset

```
+0   exinf    Extended information
+4   wtsk     Waiting task information
+6   pk_msg   Starting address of next received message
              packet
*    msgcnt   The number of messages
  *:  The offset for a 16-bit message +8
      The offset for a 32-bit message +10
```

[(Error codes)]

```
E_OK         00000H(-H'0000):      Normal End
```

[(Function description)]

Refers to the state of the mailbox specified by mbxid, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- wtsk

  wtsk returns the ID No. of the first task waiting for the specified mailbox message (the first task to start waiting). wtsk returns FALSE (0) if there are no tasks waiting for messages.

- pk_msg

  pk_msg returns the message received (the first message in the queue) when rcv_msg or trcv_msg is executed next. pk_msg returns NADR=FFFFH=(-1)[28] if there is no message.

- msgcnt

  Returns the number of messages currently in the target mailbox.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
     T_RMBX rmbx;
         :
     ref_mbx(&rmbx, ID_mbx);
         :
}
```

<< Usage example of the assembly language >>

```
 rmbx:   .blkb    10
     .INCLUDE     mr308.inc
     .GLB         task
 task:
         :
     ref_mbx      #ID_mbx,#rmbx
         :
```

[28] it returns FFFFFFFFH when message size is 32bit.

2.6. Interrupt Management Function

2.6.1. ret_int(Return from Interrupt Handler)

[(System call name)]

ret_int → Returns from the interrupt handler.

[(Calling by the C language)]

Cannot describe this system call in C language.[29]

[(Calling by the assembly language)]

```
.include mr308.inc
ret_int
```

<< Argument >>

None

<< Register setting >>

Not to return to the task which issued this system call.

[(Error codes)]

Not to return to the interrupt handler which issues a system call.

[29] If the start function of the interrupt handler is declared with #pragma INTHANDLER, the ret_int system call is automatically issued at the exit of the function.

[(Function description)]

This system call performs processing for return from the interrupt handler. Return processing involves activating the scheduler to switch over tasks as necessary.

Executing a system call in the interrupt handler does not cause task switchover to take place. Task switchover is delayed until the interrupt handler is terminated by this system call.

However, if the ret_int system call is issued from an interrupt handler that is invoked for occurrence of multiple interrupts, the scheduler is not activated. It is when the interrupt is occurred from task that the scheduler is operated.

Note that when written in the assembly language, this system call cannot be issued from a subroutine that is called from the interrupt handler entry routine. Always be sure to execute this system call in the interrupt handler entry routine or entry function.

Specifically, a program like the one below will not work normally.

```
    .include    mr308.inc
    /* NG */
    .GLB        intr
intr:
    jsr.b       func
        :
func:
    ret_int
```

To work properly, the above program must be written as follows:

```
    .include    mr308.inc
    /* OK */
    .GLB        intr
intr:
    jsr.b       func
    ret_int
func:
    :
    rts
```

This system call can only be issued from the interrupt handler. It cannot be issued from the cyclic and alarm handlers.

[(Usage example)]

<< Usage example of the C language >>

This system call cannot be written in the C language.

If the beginning function of the interrupt handler is declared with #pragma INTHANDLER,the ret_int system call is automatically issued at the exit of the function.

<< Usage example of the assembly language >>

```
    .INCLUDE mr308.inc
    .GLB        intr
intr:
        :
    iwup_tsk #ID_main
        :
    ret_int
```

2.6.2. loc_cpu(Lock CPU)

[(System call name)]

```
loc_cpu          →    Disables interrupts and task dispatch.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER loc_cpu ();
```

<< Argument >>

None

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
loc_cpu
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
```

[(Function description)]

This system call disables OS-dependent external interrupts (IPL = 0 to OS interrupt disable level) and task dispatch.

After issuing this system call, interrupts whose IPL = 0 to OS interrupt disable level and task dispatch are held in a disabled state until unl_cpu is executed. Therefore, there will be no chances that the task which has executed loc_cpu is preempted (control of CPU execution is seized) by an interrupt handler or some other task.

Interrupt requests after executing loc_cpu or dispatches generated by a system call issued by the task that has executed loc_cpu are kept pending until interrupts or dispatches are freed from the disabled condition.

If it is a task already in an interrupt or dispatch disabled state that has issued loc_cpu, no error is assumed; the result is only that the same state is continued. However, interrupt and dispatch disabled states are cleared by issuing only one unl_cpu no matter how many times loc_cpu may have been issued.

When issuing loc_cpu or dis_dsp, make sure you then issue unl_cpu or ena_dsp before ending the task (issue ext_tsk).

This system call cannot be issued from a task-independent section (e.g., interrupt handler, cyclic handler, or alarm handler).

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   loc_cpu();
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   loc_cpu
       :
```

2.6.3. unl_cpu(Unlock CPU)

[(System call name)]

unl_cpu → Enables interrupts and task dispatch.

[(Calling by the C language)]

```
#include <mr308.h>
ER unl_cpu ();
```

<< Argument >>

None

<< Return value >>

E_OK is always returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
unl_cpu
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

This system call enables external interrupts and task dispatch. Therefore, the interrupts and task dispatch that have been disabled by loc_cpu are freed from the disabled condition. Consequently, the IPL values after this system call is issued are restored to the previous IPL values when loc_cpu were issued.

If unl_cpu is issued while no interrupt and task dispatch are disabled, the system does not assume an error and only continues the same condition.

This system call cannot be issued from a task-independent section .

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
       :
   unl_cpu();
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB        task
task:
       :
   unl_cpu
       :
```

2.7. Memorypool Management Function

2.7.1. pget_blf(Poll and Get Fixed-size Memory Block)

[(System call name)]

```
pget_blf          →    Gets a fixed-size memory block(no wait)
```

[(Calling by the C language)]

```
#include <mr308.h>
ER pget_blf (p_blf,mpfid);
```

<< Argument >>

```
ID      mpfid;     The ID No. of the memory pool to be obtained
VP      *p_blf;    The start address of memory block to be obtained
```

<< Return value >>

```
An error code is returned as the return value of a function.
The start address of the obtained memory block is set to variable p_blf.
```

[(Calling by the assembly language)]

```
.include mr308.inc
pget_blf mpfid
```

<< Argument >>

```
mpfid    [---*]     The ID No. of the memory pool to be obtained
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of memory block to be obtained(lower)
R2	--
R3	The start address of memory block to be obtained(upper)
A0	The ID No. of the memory pool to be obtained

[(Error codes)]

```
E_OK        00000H(-H'0000):      Normal End
E_TMOUT     0FFABH(-H'0055):      Polling failed or timeout
```

[(Function description)]

This system call gets a memory block from the memory pool specified by mplid and returns the start address of that memory block to p_blf.

If the memory block cannot be obtained because there is no memory block in the specified memory pool, error code E_TMOUT is returned to the task which issued the system call.

Each memory block is fixed in size. The size of each memory block is defined in the configuration file. The number of memory blocks in one memory pool is up to 16.

The return parameter value(*p_blf) is undefined in all cases other than E_OK.

This system call can be issued from either tasks or handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
VP      p_blf;
void task()
{
    if( pget_blf(&p_blf,ID_mpf) != E_OK ){
        error("Not enough memory\n");
    }
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB task
p_blf: .BLKL   1
task:
        :
    pget_blf    #ID_mpf
    mov.w       R1, p_blf
    mov.w       R3, p_blf+2
        :
    ext_tsk
```

2.7.2. rel_blf(Release Fixed-size Memory Block)

[(System call name)]

```
rel_blf          →    Release a fixed-size memory block
```

[(Calling by the C language)]

```
#include <mr308.h>
ER rel_blf (mpfid, p_blf);
```

<< Argument >>

```
ID       mpfid;     The ID No. of the memory pool to be released
VP       p_blf;     The start address of memory block to be release
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
rel_blf  mpfid, p_blf
```

<< Argument >>

```
mpfid    [−−−*]     The ID No. of the memory pool to be released
p_blf    [−***]     The start address of memory block to be release
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of memory block to be release(lower)
R2	--
R3	The start address of memory block to be release(upper)
A0	The ID No. of the memory pool to be released

[(Error codes)]

```
E_OK            00000H(-H'0000):        Normal End
```

[(Function description)]

This system call returns the memory block whose start address is specified by p_blf to the memory pool. For the start address of the memory block to be freed (returned), always use the value obtained by pget_blf.

This system call does not especially check whether p_blf is pointing at the start address of the correct memory block.

This system call can be issued from either tasks or handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
#define ID_mpf1 1
void task()
{
    VP   p_blf;
    if( pget_blf(&p_blf, ID_mpf1) != E_OK )
        error("Not enough memory \n");
        :
    rel_blf(ID_mpf1,p_blf);
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        _task
p_blf:  .BLKL   1

_task:
        :
    pget_blf    #ID_mpf1
    mov.w       R1, p_blf
    mov.w       R3, p_blf+2
        :
        :
    rel_blf     #ID_mpf1,p_blf
        :
    ext_tsk
```

2.7.3. ref_mpf(Refer Fixed-size Memorypool Status)

[(System call name)]

ref_mpf → Reference Fixed-size Memorypool Status

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_mpf (pk_rmpf, mpfid);
```

<< Argument >>

```
T_RMPF  *pk_rmpf;   Packet address to Reference fixed-size
                    memorypool
ID      mpfid;      The ID No. of the memorypool to Reference
                    fixed-size memorypool
```

<< Return value >>

An error code is returned as the return value of a function.
The structure indicated by pk_rmpf returns the following data.

```
typedef  struct t_rmpf {
    VP      exinf;  /* Extended information */
    BOOL_ID wtsk;   /* Waiting task information */
    INT     frbcnt; /* The number of free blocks */
    INT     blksz;  /* The size of blocks */
} T_RMPF;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_mpf  mpfid, pk_rmpf
```

<< Argument >>

```
mpfid    [---*]   The ID No. of the memorypool to Reference
                  fixed-size memorypool
pk_rmpf  [-***]   Packet address to Reference fixed-size
                  memorypool
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the memorypool to Reference fixed-size memorypool
A1	Packet address to Reference fixed-size memorypool

The structure indicated by pk_rmpf returns the following data

offset

```
+0   exinf    Extended information
+4   wtsk     Waiting task information
+6   frbcnt   The number of free blocks
+8   blksz    The size of blocks
```

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

Refers to the state of the fixed-size memorypool specified by mpfid, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- wtsk

  wtsk returns the ID No. of the first task waiting for the specified memorypool. In the MR308, however, wtsk always returns FALSE(0), because tasks cannot enter the wait state for the memorypool.

- frbcnt

  Returns the number of free blocks in the specified fixed-size memorypool.

- blksz

  Returns the size of blocks in the specified fixed-size memorypool.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    T_RMPF rmpf;
        :
    ref_mpf(ID_mpf, &rmpf );
        :
    ext_tsk();
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
rmpf:   .BLKL   8

task:
        :
    ref_mpf     #ID_mpf,#rmpf
        :
```

2.7.4. pget_blk(Poll and Get Variable-size Memory Block)

[(System call name)]

```
pget_blk          →    Gets a variable-size memory block(no wait)
```

[(Calling by the C language)]

```
#include <mr308.h>
ER pget_blk (p_blk,mplid,blksz);
```

<< Argument >>

```
VP       *p_blk;    The start address of memory block to be obtained
ID       mplid;     The ID No. of the memory block to be obtained
INT      blksz;     Memory block size to be obtained
```

<< Return value >>

```
An error code is returned as the return value of a function.
The start address of the obtained memory block is set to variable p_blk.
```

[(Calling by the assembly language)]

```
.include mr308.inc
pget_blk blksz
```

<< Argument >>

```
blksz    [—**]      Memory block size to be obtained
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of memory block to be obtained(lowe)
R2	--
R3	The start address of memory block to be obtained(upper)
A0	--

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
E_TMOUT       0FFABH(-H'0055):      Polling failed or timeout
```

[(Function description)]

This system call gets a variable-size memory block from the memorypool specified by mplid and returns the start address of that memory block to p_blk.

If the memory block cannot be obtained because there is no memory block in the specified memorypool, error code E_TMOUT is returned to the task which issued the system call.

MR308 support 1 memorypool. So, you must specify mplid as 1.

The size of each memory block is defined in the configuration file.

The return parameter value(*p_blk) is undefined in all cases other than E_OK.

This system call can be issued only from tasks.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
#define ID_mpl1 1
VP      p_blk;
void task()
{
    /* Get 70 bytes memory block */
    if( pget_blk(&p_blk,ID_mpl1,70) != E_OK )-
        error("Not enough memory\n");
    }
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE mr308.inc
    .GLB task

p_blk: .blkb 4
task:
        :
    pget_blk    #50             ; Get 50 bytes memory block
    mov.w       R1, p_blk
    mov.w       R3, p_blk+2
        :
    ext_tsk
```

2.7.5. rel_blk(Release Variable-size Memory Block)

[(System call name)]

```
rel_blk          →    Release a variable-size memory block
```

[(Calling by the C language)]

```
#include <mr308.h>
ER rel_blk (mplid,p_blk);
```

<< Argument >>

```
ID      mplid;    The ID No. of the memory block to be released
VP      p_blk;    The start address of memory block to be release
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
rel_blk  p_blk
```

<< Argument >>

```
p_blk    [-***]     The start address of memory block to be release
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	The start address of memory block to be release(lower)
R2	--
R3	The start address of memory block to be release(upper)
A0	--

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

This system call returns the memory block whose start address is specified by p_blk to the memorypool.

For the start address of the memory block to be freed (returned), always use the value obtained by pget_blk.

This system call does not especially check whether p_blk is pointing at the start address of the correct memory block. This system call can be issued only from tasks .

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
#define ID_mpl1 1
void task()
{
    VP   p_blk;
    /* Get 60 bytes memory block */
    if( pget_blk(&p_blk,ID_mpl1,60) != E_OK )
        error("Not enough memory \n");
        :
        :
    rel_blk(ID_mpl1,p_blk);     /* Release memory block */
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        _task

    p_blk:      .BLKL   1
_task:
        :
    pget_blk    #60             ; Get 60 bytes memory block
    mov.w       R1, p_blk
    mov.w       R3, p_blk+2
        :
    rel_blk     p_blk           ; Release memory block
```

2.7.6. ref_mpl(Refer Variable-size Memorypool Status)

[(System call name)]

```
ref_mpl          →    Reference Variable-size Memorypool Status
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_mpl (pk_rmpl, mplid);
```

<< Argument >>

```
T_RMPL  *pk_rmpl;   Packet address to Reference variable-size
                    memorypool
ID      mplid;      The ID No. of the memory block to Reference
                    variable-size memorypool(only 1)
```

<< Return value >>

```
An error code is returned as the return value of a function.
The structure indicated by pk_rmpl returns the following data.
 typedef struct t_rmpl {
    VP      exinf;  /* Extended information */
    BOOL_ID wtsk;   /* Waiting task information */
    W       frsz;   /* The total size of the free area */
    INT     maxsz;  /* The size of the maximum free area */
 } T_RMPL;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_mpl  pk_rmpl
```

<< Argument >>

```
pk_rmpl  [-***]     Packet address to Reference variable-size
                    memorypool
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--
A1	Packet address to Reference variable-size memorypool

The structure indicated by pk_rmpl returns the following data

offset

```
+0    exinf     Extended information
+4    wtsk      Waiting task information
+6    frsz      The total size of the free area
+10   maxsz     The size of the maximum free area
```

[(Error codes)]

```
E_OK           00000H(-H'0000):       Normal End
```

[(Function description)]

Refers to the state of the variable-size memorypool specified by mplid, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- wtsk

  wtsk returns the ID No. of the first task waiting for the specified variable-size memory-pool. In the MR308, however, wtsk always returns FALSE (0), because tasks cannot enter the wait state for the memorypool.

- frsz

  Returns the total size of the free area.

- maxsz

  Returns the size of the maximum free area that can immediately be obtained.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    T_RMPL rmpl;
    ref_mpl(&rmpl,ID_mpl1);
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
rmpl:   .BLKB   10

task:
        :
    ref_mpl     #rmpl
        :
```

2.8. Time Management Function

2.8.1. set_tim(Set Time)

[(System call name)]

```
set_tim          →    Sets the system clock.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER set_tim (pk_tim);
```

<< Argument >>

```
SYSTIME   *pk_tim;   The start address of packet specifying the
                     system clock to be set
```

<< Return value >>

```
E_OK is always returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
set_tim  pk_tim
```

<< Argument >>

```
pk_tim   [-***]      The start address of packet specifying the
                     system clock to be set
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The start address of packet specifying the system clock to be set

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

This system call sets the value of the system clock to a value indicated by pk_tim. [30]

The 48-bit system clock is handled separately in ltime, mtime, and utime.

If a system time is reset (etc. set_tim () system call) to a time before an Alarm Handler already started, the Alarm Handler will never restart. If the system time is reset to a time after an Alarm Handler starts, all Alarm Handler will never start.

This system call can be issued from both tasks and handlers.

[30] The system time is 0 when the system is reset, and the number of system clock interrupts generated is indicated by 48-bit data.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    SYSTITIME time;     /* Time data storing variable */
    time.utime = 2;     /* Sets upper time data */
    time.mtime = 1;     /* Sets middle time data */
    time.ltime = 0;     /* Sets lower time data */
    set_tim( &time );
}
```

<< Usage example of the assembly language >>

```
     .INCLUDE    mr308.inc
     .GLB        task
time:
     .WORD       2
     .WORD       1
     .WORD       0

 task:
     set_tim     #time
         :
```

2.8.2. get_tim(Get Time)

[(System call name)]

```
get_tim          →   Reads the system clock value.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER get_tim (pk_tim);
```

<< Argument >>

```
SYSTIME   *pk_tim;   The start address of packet in which the read
                     system clock is stored
```

<< Return value >>

```
E_OK is always returned as the return value of a function.
The current time data is returned to the structure which pk_tim is
specifying.
```

[(Calling by the assembly language)]

```
.include mr308.inc
get_tim
```

<< Argument >>

```
pk_tim   [−***]      The start address of packet in which the read
                     system clock is stored
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The start address of packet in which the read system clock is stored

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

This system call reads out the current value of the system clock and returns it to return parameter pk_tim. [31]

The 48-bit system clock time is handled separately in ltime, mtime, and utime.

This system call can be issued from both tasks and handlers.

[31] The system time is 0 at reset. The number of times the system clock interrupt occurred is represented in 48-bit data.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
 #include "id.h"
 void task()
 {
     SYSTIME     time;            /* Time data storing variable */
     get_tim( &time );            /* Reads system time */
     printf("system_clock.utime = %X\n",time.utime);
     printf("system_clock.mtime = %X\n",time.mtime);
     printf("system_clock.ltime = %X\n",time.ltime);
 }
```

<< Usage example of the assembly language >>

```
     .INCLUDE    mr308.inc
     .GLB        task
time:
     .BLKW       3

 task:
     get_tim     #time
         :
```

2.8.3. dly_tsk(Delay Task)

[(System call name)]

```
dly_tsk           →    Delays task execution.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER dly_tsk (dlytim);
```

<< Argument >>

```
DLYTIME  dlytim;    Delay time
```

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
dly_tsk  dlytim
```

<< Argument >>

```
dlytim   [--**]      Delay time
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	Delay time
A0	--

[(Error codes)]

```
E_OK           00000H(-H'0000):        Normal End
E_RLWAI        0FFAAH(-H'0056):        Wait state forcibly cleared
```

[(Function description)]

This system call temporarily stops execution of the own task for a duration specified by dlytim, with the task placed from the execution (RUN) state into a wait (WAIT) state.

A wait state invoked by this system call is cancelled in the following cases:

When the wait state is cancelled, the task that invoked this system call exits from the timeout wait queues and is connected to the ready queue.

- When the time specified in dlytim has elapsed.

  Error code E_OK is returned.

- When the wait state is forcibly cancelled by rel_wai or irel_wai system calls before the dlytim time has elapsed.

  Error code E_RLWAI is returned.

However, the wait state is not cleared by executing iwup_tsk or wup_tsk during a delay.

The unit of time specified in dlytim is the unit of time of the system clock, specified in the configuration file.[32]

The maximum value of dlytim is 0x7FFF (32,767).

dly_tsk(5);

For example, if it is 10ms and the following is written in the program the own task is placed from the execution (RUN) state into a wait (WAIT) state and held in that state for 50 ms.

This system call can be issued only from tasks. It cannot be issued from the in terrupt handler, the cyclic handler, or the alarm handler.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    if( dly_tsk( 10 ) != E_OK );
        printf("Forced wakeup\n");
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
 task:
    dly_tsk     #200
        :
```

[32] Refer Users Manual how to specify the unit of time of the system clock in the configuration file.

2.8.4. act_cyc (Activate Cyclic Handler)

[(System call name)]

```
act_cyc          →    Controls the activation of the cyclic handler.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER act_cyc (cycno, cycact);
```

<< Argument >>

```
HNO      cycno;     The cyclic handler specification number
UINT     cycact;    The cyclic handler activation status
```

<< Return value >>

```
E_OK is always returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
act_cyc  cycno, cycact
```

<< Argument >>

```
cycno     [−−−*]     The cyclic handler specification number
cycact    [−−**]     The cyclic handler activation status
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	The cyclic handler activation status
A0	The cyclic handler specification number

[(Error codes)]

```
E_OK          00000H(-H'0000):       Normal End
```

[(Function description)]

This system call changes the activation status of the cyclic handler specified by cyhno.

That is, it enables or disables the cyclic handler.

The following three specifications can be made by cyhact:

Specifications of Cyclic Handler Activation Status

C language	Assembly language	Meaning
TCY_OFF	TCY_OFF	Disables the cyclic handler
TCY_ON	TCY_ON	Enables the cyclic handler
TCY_ON|TCY_INI	TCY_INI_ON	Enables the cyclic handler and|clears the cyclic counter at the same|time.

The cyclic handler is executed as a part of the system clock interrupt handler.[33]

This system call can be issued from both tasks and handlers.

[33] Namely, the cyclic handler is called from the system clock handler by a subroutine call.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
        :
    act_cyc ( ID_cyc, TCY_ON );
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
    act_cyc     #ID_cyc, #TCY_INI_ON
        :
```

2.8.5. ref_cyc(Refer Cyclic Handler Status)

[(System call name)]

```
ref_cyc          →    Reference Cyclic handler Status
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_cyc (pk_rcyc, cycno);
```

<< Argument >>

```
HNO      cycno;      The cyclic handler specification number
T_RCYC   *pk_rcyc;   Packet address to Reference cyclic handler
```

<< Return value >>

```
An error code is returned as the return value of a function.
The structure indicated by pk_rcyc returns the following data.
 typedef struct t_rcyc {
    VP      exinf;  /* Extended information */
    CYCTIME lfttim; /* The time remaining until the next cycle start
                       handler starts */
    UINT    cycact; /* The active state of the cycle start handler */
 }T_RCYC;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_cyc  cycno, pk_rcyc
```

<< Argument >>

```
cycno     [---*]     The cyclic handler specification number
pk_rcyc   [-***]     Packet address to Reference cyclic handler
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The cyclic handler specification number
A1	Packet address to Reference cyclic handler

The structure indicated by pk_rcyc returns the following data.

Offset

```
+0    exinf     Extended information
+4    lftim     The time remaining until the next cycle start
                handler starts
+6    cycact    The active state of the cycle start handler
```

[(Error codes)]

```
E_OK            00000H(-H'0000):       Normal End
```

[(Function description)]

Refers to the state of the cyclic handler specified by almno, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- lfttime

  lftim returns the time remaining until the next cycle start handler starts. The time remaining until the next cycle start handler starts is expressed as the number of system clock counts.

- cycact

  cycact returns the active state of the cycle start handler. That is, cycact returns TCY_ON (=1) when the cycle start handler is ON, and TCY_OFF (=0) when it is OFF.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
    T_RCYC rcyc;
    ref_cyc( &rcyc, ID_cyc );
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task
task:
         :
    ref_cyc     #ID_cyc,#rcyc
         :
```

2.8.6. ref_alm(Refer Alarm Handler Status)

[(System call name)]

```
ref_alm          →    Reference Alarm handler Status
```

[(Calling by the C language)]

```
#include <mr308.h>
ER ref_alm (pk_ralm, almno);
```

<< Argument >>

```
HNO     almno;      The alarm handler specification number
T_RALM  *pk_ralm;   Packet address to Reference alarm handler
```

<< Return value >>

```
An error code is returned as the return value of a function.
The structure indicated by pk_ralm returns the following data.
 typedef struct t_ralm {
     VP      exinf;  /* Extended information */
     ALMTIME lfttim; /* The time remaining until the next alarm start
                        handler starts */
 }T_RALM;
```

[(Calling by the assembly language)]

```
.include mr308.inc
ref_alm  almno, pk_ralm
```

<< Argument >>

```
almno     [---*]      The alarm handler specification number
pk_ralm   [-***]      Packet address to Reference alarm handler
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	Packet address to Reference alarm handler
A1	The alarm handler specification number

The structure indicated by pk_ralm returns the following data.

offset

```
+0   exinf     Extended information
+4   lftim     The time remaining until the next alarm start
               handler starts
```

[(Error codes)]

```
E_OK            00000H(-H'0000):       Normal End
```

[(Function description)]

Refers to the state of the alarm handler specified by almno, and returns returns the following information as return values.

- exinf

  Returns extended task information in exinf(Always indeterminate value)

- lfttim

  lfttim returns the time remaining until the specified alarm handler is started. The time remaining until the alarm handler starts is expressed as 48-bit data showing the number of times the system clock interrupt remains to be invoked.

  The 48-bit system time is divided into ltime, mtime, and utime.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void func()
{
   T_RALM   ralm;
   ref_alm( &ralm, ID_alarm );
       :
}
```

<< Usage example of the assembly language >>

```
   .INCLUDE    mr308.inc
   .GLB task
task:
   ref_alm     #ID_alm,#ralm
       :
```

2.9. Version Management Function

2.9.1. get_ver(Get Version Information)

[(System call name)]

```
get_ver          →    Gets the version number of the MR308.
```

[(Calling by the C language)]

```
#include <mr308.h>
ER get_ver (pk_ver);
```

<< Argument >>

```
T_VER    *pk_ver;   The start address of the structure in which
                    version information is stored
```

<< Return value >>

```
E_OK is always returned as the return value of a function.
The version information is set to structure pk_ver.
```

[(Calling by the assembly language)]

```
.include mr308.inc
get_ver  pk_ver
```

<< Argument >>

```
pk_ver   [−***]     The start address of the structure in which
                    version information is stored
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The start address of the structure in which version information is stored

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
```

[(Function description)]

This system call gets the version number and other information on the MR308.

The version number is obtained in the format standardized by the TRON specifications.

Therefore, the version number can be obtained in the format common to different types of microcomputers or the operating systems of different TRON specifications.

The version information can be obtained is as follows:

```
UH   maker      /* Maker */
UH   id         /* Format number */
UH   spver      /* Specification version */
UH   prver      /* Product version */
UH   prno[4]    /* Product control information */
UH   cpu        /* CPU information */
UH   var        /* Variation descriptor*/
```

The version No. formats are as follows:

1. Maker
   H'0115 indicating Renesas Technology Corporation is returned.

2. Format number
   Internal identification ID H'150 of the MR308 is returned.
3. Specification version
   H'5302 indicating the μITRONspecifications Ver.3.02 is returned.
4. Product version
   H'110 indicating the version of the MR308 is returned.
5. Product control information
   - prno[0]

     The product release number is obtained

     prno[0] ← 0001H

   - prno[1]

     The product release year and month are obtained

     prno[1] ← 0007H

   - prno[2]

     Reserved for Renesas use.

     prno[2] ← ????H

   - prno[3]

     Reserved for Renesas use.

     prno[3] ← ????H
6. CPU information
   H'C25 indicating the M16C/80 Series Micro computeris returned.
7. Variation descriptor
   H'8000 indicating the variation of the MR308 is returned.

This system call can be issued from both tasks and handlers.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task()
{
   T_VER   pk_ver;
   get_ver( &pk_ver );
}
```

<< Usage example of the assembly language >>

```
ver:
   .BLKW       10
   .INCLUDE    mr308.inc
   .GLB        task
task:
   get_ver     #ver
       :
```

2.10. Implementation-Dependent System Call

2.10.1. vrst_msg(Reset Message)

[(System call name)]

```
vrst_msg        →    Clears messages in mailbox
```

[(Calling by the C language)]

```
#include <mr308.h>
ER vrst_msg ( mbxid );
```

<< Argument >>

```
ID      mbxid;    The ID No. of the mailbox to clear Mailbox
```

<< Return value >>

```
An error code is returned as the return value of a function.
```

[(Calling by the assembly language)]

```
.include mr308.inc
vrst_msg mbxid
```

<< Argument >>

```
mbxid    [---*]    The ID No. of the mailbox to clear Mailbox
```

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
R3	--
A0	The ID No. of the mailbox to clear Mailbox

[(Error codes)]

```
E_OK          00000H(-H'0000):      Normal End
```

[(Function description)]

Clears the messages stored in the mailbox specified in mbxid. If there are no messages in the specified mailbox, no processing is performed.

E_OK is always returned as the return value of a function.

This system call can be issued only from tasks.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task1(void)
{
    ER ercd;
        :
    ercd = vrst_msg( ID_mbx );
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task1
task1:
        :
    vrst_msg #ID_mbx
        :
    ext_tsk
```

2.10.2. vrst_blf(Reset Fixed-Memory Block)

[(System call name)]

vrst_blf → Releases all specified fixed-size memory blocks

[(Calling by the C language)]

```
#include <mr308.h>
ER vrst_blf ( mbfid );
```

<< Argument >>

ID mpfid; The ID No. of the memorypool to release

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
vrst_blf mpfid
```

<< Argument >>

mpfid [−−−*] The ID No. of the memorypool to release

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	The ID No. of the memorypool to release

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

Releases all memory blocks with the specified fixed-size memorypool ID No.

If the specified memorypool has not been used by any task, no processing is performed when this system call is invoked.

This system call can be issued only from tasks.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task1(void)
{
    ER ercd;
        :
    ercd = vrst_blf( ID_mpf );
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task1
task1:
        :
    vrst_blf #ID_mpf
        :
    ext_tsk
```

2.10.3. vrst_blk(Reset Variable-Memory Block)

[(System call name)]

vrst_blk → Releases all specified variable-size memory blocks

[(Calling by the C language)]

```
#include <mr308.h>
ER vrst_blk ( void );
```

<< Argument >>

None

<< Return value >>

An error code is returned as the return value of a function.

[(Calling by the assembly language)]

```
.include mr308.inc
vrst_blk
```

<< Argument >>

None

<< Register setting >>

Register name	Contents after system call issuance
R0	Error code
R1	--
R2	--
A0	--

[(Error codes)]

E_OK 00000H(-H'0000): Normal End

[(Function description)]

Releases all memory blocks with the specified variable-size memorypool ID No.

If the specified memorypool has not been used by any task, no processing is performed when this system call is invoked.

This system call can be issued only from tasks.

[(Usage example)]

<< Usage example of the C language >>

```
#include <mr308.h>
#include "id.h"
void task1(void)
{
    ER ercd;
        :
    ercd = vrst_blk();
        :
}
```

<< Usage example of the assembly language >>

```
    .INCLUDE    mr308.inc
    .GLB        task1
task1:
        :
    vrst_blk
        :
    ext_tsk
```

Chapter 3 Appendix

3.1. List of System calls

Task Manegement Functions

System call		Function	Scheduler
sta_tsk	[S]	Starts a task.	call
ista_tsk	[S]	Starts a task.(handler only)	--
ext_tsk	[S]	Normally ends the self task.	call
ter_tsk	[S]	Forcibly ends other task.	call
chg_pri	[S]	Changes the task priority.	call
ichg_pri	[S]	Changes the task priority.(handler only)	--
dis_dsp	[S]	Disables task dispatch.	--
ena_dsp	[S]	Enables task dispatch.	call
rot_rdq	[S]	Rotates the task ready queue.	call
irot_rdq	[S]	Rotates the task ready queue.(handler only)	--
rel_wai	[S]	Forcibly clears the task wait state.	call
irel_wai	[S]	Forcibly clears the task wait state.(handler only)	--
get_tid	[S]	Gets the ID of self task.	--
ref_tsk	[E]	Reference Task Status.	--

Synchronization Functions Attached to Task

System call		Function	Scheduler
sus_tsk	[S]	Puts a task into the suspend state.	call
isus_tsk	[S]	Puts a task into the suspend state.(handler only)	--
rsm_tsk	[S]	Resumes the suspended task.	call
irsm_tsk	[S]	Resumes the suspended task.(handler only)	--
slp_tsk	[R]	Puts a task into the wait state.	call
tslp_tsk	[E]	Puts a task into the wait state.(With Timeout)	call
wup_tsk	[R]	Wakes up the waiting task.	call
iwup_tsk	[R]	Wakes up the waiting task.(handler only)	--
can_wup	[S]	Cancels the request for waking up a task	--

Synchronization and Communication Functions

System call		Function	Scheduler
set_flg	[S]	Sets an event flag.	call
iset_flg	[S]	Sets an event flag.(handler only)	--
clr_flg	[S]	Clears an event flag.	--
wai_flg	[S]	Waits for an event flag.	call
twai_flg	[E]	Waits for an event flag. (With Timeout)	call
pol_flg	[S]	Gets an event flag. (no wait)	--
ref_flg	[E]	Reference Eventflag Status.	--
sig_sem	[R]	Signal operation for a semaphore	call
isig_sem	[R]	Signal operation for a semaphore. (handler only)	--
wai_sem	[R]	Wait operation for a semaphore.	call
twai_sem	[E]	Wait operation for a semaphore. (With Timeout)	call
preq_sem	[R]	Gets the semaphore resource. (no wait)	--
ref_sem	[E]	Reference Semaphore Status.	--
snd_msg	[S]	Sends a message.	call
isnd_msg	[S]	Sends a message (handler only).	--
rcv_msg	[S]	Waits for message reception.	call
trcv_msg	[E]	Waits for message reception. (With Timeout)	call
prcv_msg	[S]	Receives a message.(no wait)	--
ref_mbx	[E]	Reference Mailbox Status.	--

Interrupt Management Functions

System call		Function	Scheduler
ret_int	[R]	Returns from the interrupt handler.	call
loc_cpu	[R]	Disables OS-dependent interrupt and task dispatch.	--
unl_cpu	[R]	Enables OS -dependent interrupt and task dispatch.	call

Memorypool Management Functions

System call		Function	Scheduler
pget_blf	[E]	Gets fixed-size memory block	--
rel_blf	[E]	Release fixed-size memory block.	--
ref_mpf	[E]	Reference fixed-size Memorypool status.	--
pget_blk	[E]	Gets variable-size memory|block.	call
rel_blk	[E]	Release variable-size memory|block.	call
ref_mpl	[E]	Reference variable-size Memorypool status.	--

TIme Management Functions

System call		Function	Scheduler
set_tim	[S]	Sets the system clock.	--
get_tim	[S]	Reads the system clock value.	--
dly_tsk	[S]	Delays the task.	call
act_cyc	[E]	Controls activation of the cyclic handler.	--
ref_cyc	[E]	Reference Cyclic handler Status.	--
ref_alm	[E]	Reference Alarm Handler Status.	--

Version Management Function

System call		Function	Scheduler
get_ver	[R]	Gets the OS version number.	--

Implementation-Dependent System Call

System call		Function	Scheduler
vrst_msg	[--]	Cears messages in mailbox	--
vrst_blf	[--]	Releases all specified fixed-size memory blocks	--
vrst_blk	[--]	Releases all specified valiable-size memory blocks	--

3.2. List of Error code

Error code	Value	Description
E_OK	00000H(-H'0000)	Normal End
E_OBJ	0FFC1H(-H'003F)	Invalid object state
E_QOVR	0FFB7H(-H'0049)	Queuing or nest overflow
E_TMOUT	0FFABH(-H'0055)	Polling failed or timeout
E_RLWAI	0FFAAH(-H'0056)	Wait state forcibly cleared

3.3. Assembly Language Interface

When issuing a system call in the assembly language, you need to use macros prepared for invoking system calls.

Processing in a system call invocation macro involves setting each parameter to registers and starting execution of a system call routine by a software interrupt.

If you issue system calls directly without using a system call invocation macro, your program may not be guaranteed of compatibility with future versions of MR308. The table below lists the assembly language interface parameters. The values set forth in μITRON specifications are not used for the function code.

Task Manegement Functions

System-call	INT No.	Parameter				Return Parameter		
		R0 (Function code)	R1	R2	A0	R0	R2	A0
sta_tsk	#63	H'00	-	stacd	tskid	ercd	-	-
ista_tsk	#62	H'24	-	stacd	tskid	ercd	-	-
ext_tsk	#58	-	-	-	-	-	-	-
ter_tsk	#63	H'02	-	-	tskid	ercd	-	-
dis_dsp	#60	-	-	-	-	ercd	-	-
ena_dsp	#63	H'0a	-	-	-	ercd	-	-
chg_pri	#63	H'04	-	tskpri	tskid	ercd	-	-
ichg_pri	#62	H'26	-	tskpri	tskid	ercd	-	-
rot_rdq	#63	H'06	-	tskpri	-	ercd	-	-
irot_rdq	#62	H'28	-	tskpri	-	ercd	-	-
rel_wai	#63	H'08	-	-	tskid	ercd	-	-
irel_wai	#62	H'2a	-	-	tskid	ercd	-	-
get_tid	#62	H'2c	-	-	-	ercd	-	tskid

System-call	INT No.	Parameter			Return Parameter		
		R0 (Function code)	A0	A1	R0	R2	A0
ref_tsk	#62	H'54	tskid	pk_rtsk	ercd	-	-

Synchronization Functions Attached to Task

System-call	INT No.	Parameter				Return Parameter		
		R0 (Function code)	R1	R2	A0	R0	R2	A0
sus_tsk	#63	H'0c	-	-	tskid	ercd	-	-
isus_tsk	#62	H'2e	-	-	tskid	ercd	-	-
rsm_tsk	#63	H'0e	-	-	tskid	ercd	-	-
irsm_tsk	#62	H'30	-	-	tskid	ercd	-	-
slp_tsk	#63	H'10	-	-	-	ercd	-	-
wup_tsk	#63	H'12	-	-	tskid	ercd	-	-
iwup_tsk	#62	H'32	-	-	tskid	ercd	-	-
can_wup	#62	H'34	-	-	tskid	ercd	wupcnt	-

System-call	INT No.	Parameter					Return Parameter		
		R0 (Function code)	R1	R2	R3	A0	R0	R1	R2
tslp_tsk	#63	H'64	-	-	tmout	-	ercd	-	-

Synchronization and Communication Functions

System-call	INT No.	Parameter				Return Parameter		
		R0 (Function code)	R1	R2	A0	R0	R2	A0
set_flg	#63	H'14	-	setptn	flgid	ercd	-	-
iset_flg	#62	H'36	-	setptn	flgid	ercd	-	-
clr_flg	#63	H'38	-	clrptn	flgid	ercd	-	-
wai_flg	#63	H'16	wfmode	waiptn	flgid	ercd	flgptn	-
pol_flg	#62	H'3a	wfmode	waiptn	flgid	ercd	flgptn	-
sig_sem	#63	H'18	-	-	semid	ercd	-	-
isig_sem	#62	H'3c	-	-	semid	ercd	-	-
wai_sem	#63	H'1a	-	-	semid	ercd	-	-
preq_sem	#62	H'3e	-	-	semid	ercd	-	-
snd_msg	#63	H'1c	-	pk_msg	mbxid	ercd	-	-
isnd_msg	#62	H'40	-	pk_msg	mbxid	ercd	-	-
rcv_msg	#63	H'1e	-	-	mbxid	ercd	pk_msg	-
prcv_msg	#62	H'42	-	-	mbxid	ercd	pk_msg	-

System-call	INT No.	Parameter					Return Parameter		
		R0 (Function code)	R1	R2	R3	A0	R0	R1	R2
twai_flg	#63	H'66	wfmode	waiptn	tmout	flgid	ercd	-	flgptn
twai_sem	#63	H'68	-	-	tmout	semid	ercd	-	-
trcv_msg	#63	H'6a	tmout	-	-	mbxid	ercd	-	pk_msg
trcv_msg*	#63	H'6a	tmout	-	-	mbxid	ercd	pk_msg	pk_msg

*: 32bit message size

System-call	INT No.	Parameter			Return Parameter		
		R0 (Function code)	A0	A1	R0	R2	A0
ref_flg	#62	H'56	flgid	pk_rflg	ercd	-	-
ref_sem	#62	H'58	semid	pk_rsem	ercd	-	-
ref_mbx	#62	H'5a	mbxid	pk_rmbx	ercd	-	-

System-call	INT No.	Parameter				Return Parameter			
		R0 (Function code)	R1	R3	A0	R0	R1	R2	R3
snd_msg*	#63	H'1c	pk_msg	pk_msg	mbxid	ercd	-	-	-
isnd_msg*	#62	H'40	pk_msg	pk_msg	mbxid	ercd	-	-	-
rcv_msg*	#63	H'1e	-	-	mbxid	ercd	pk_msg	pk_msg	-
prcv_msg*	#62	H'42	-	-	mbxid	ercd	pk_msg	pk_msg	-

*: 32bit message size

Interrupt Management Functions

System-call	INT No.	Parameter				Return Parameter		
		R0 (Function code)	R1	R2	A0	R0	R2	A0
ret_int	#61	-	-	-	-	-	-	-
loc_cpu	#59	-	-	-	-	ercd	-	-
unl_cpu	#63	H'20	-	-	-	ercd	-	-

Memorypool Management Functions

System-call	INT No.	Parameter				Return Parameter			
		R0 (Function code)	R1	R3	A0	R0	R1	R2	R3
pget_blf	#62	H'4c	-	-	mpfid	ercd	p_blf	-	p_blf
rel_blf	#62	H'4e	p_blf	p_blf	mpfid	ercd	-	-	-
pget_blk	#63	H'50	-	-	-	ercd	p_blk	-	p_blk
rel_blk	#63	H'52	p_blk	p_blk	-	ercd	-	-	-

System-call	INT No.	Parameter			Return Parameter		
		R0 (Function code)	A0	A1	R0	R2	A0
ref_mpf	#62	H'5c	mpfid	pk_rmpf	ercd	-	-
ref_mpl	#62	H'5e	-	pk_rmpl	ercd	-	-

TIme Management Functions

System-call	INT No.	Parameter				Return Parameter		
		R0 (Function code)	R1	R2	A0	R0	R2	A0
set_tim	#62	H'44	-	-	pk_tim	ercd	-	-
get_tim	#62	H'46	-	-	pk_tim	ercd	-	-
dly_tsk	#63	H'22	-	dlytim	-	ercd	-	-
act_cyc	#62	H'48	-	cycact	cycno	ercd	-	-

System-call	INT No.	Parameter			Return Parameter		
		R0 (Function code)	A0	A1	R0	R2	A0
ref_cyc	#62	H'60	cycno	pk_rcyc	ercd	-	-
ref_alm	#62	H'62	almno	pk_ralm	ercd	-	-

Version Management Function

System-call	INT No.	Parameter				Return Parameter		
		R0 (Function code)	R1	R2	A0	R0	R2	A0
get_ver	#62	H'4a	-	-	pk_ver	ercd	-	-

Implementation-Dependent System Call

System-call	INT No.	Parameter			Return Parameter		
		R0 (Function code)	A0	A1	R0	R2	A0
vrat_msg	#55	H'0	mbxid	-	ercd	-	-
vrst_blf	#55	H'2	mpfid	-	ercd	-	-
vrst_blk	#55	H'4	-	-	ercd	-	-

3.4. C Language Interface

Task Manegement Functions

ER	ercd =	sta_tsk	(ID tskid, INT stacd);
ER	ercd =	ista_tsk	(ID tskid, INT stacd);
	void	ext_tsk	();
ER	ercd =	ter_tsk	(ID tskid);
ER	ercd =	dis_dsp	();
ER	ercd =	ena_dsp	();
ER	ercd =	chg_pri	(ID tskid, PRI tskpri);
ER	ercd =	ichg_pri	(ID tskid, PRI tskpri);
ER	ercd =	rot_rdq	(PRI tskpri);
ER	ercd =	irot_rdq	(PRI tskpri);
ER	ercd =	rel_wai	(ID tskid);
ER	ercd =	irel_wai	(ID tskid);
ER	ercd =	get_tid	(ID *p_tskid);
ER	ercd =	ref_tsk	(T_RTSK *pk_rtsk, ID tskid);

Synchronization Functions Attached to Task

ER	ercd =	sus_tsk	(ID tskid);
ER	ercd =	isus_tsk	(ID tskid);
ER	ercd =	rsm_tsk	(ID tskid);
ER	ercd =	irsm_tsk	(ID tskid);
ER	ercd =	slp_tsk	();
ER	ercd =	tslp_tsk	(TMO tmout);
ER	ercd =	wup_tsk	(ID tskid);
ER	ercd =	iwup_tsk	(ID tskid);
ER	ercd =	can_wup	(INT *p_wupcnt, ID tskid)

Synchronization and Communication Functions

ER	ercd =	set_flg	(ID flgid, UINT setptn);
ER	ercd =	iset_flg	(ID flgid, UINT setptn);
ER	ercd =	clr_flg	(ID flgid, UINT clrptn);
ER	ercd =	wai_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);
ER	ercd =	pol_flg	(UINT *p_flgptn, ID flgid, UINT waiptn, UINT wfmode);
ER	ercd =	ref_flg	(T_RFLG *pk_rflg, ID flgid);
ER	ercd =	sig_sem	(ID semid);
ER	ercd =	isig_sem	(ID semid);
ER	ercd =	wai_sem	(ID semid);
ER	ercd =	twai_sem	(ID semid, TMO tmout);
ER	ercd =	preq_se m	(ID semid);
ER	ercd =	ref_sem	(T_RSEM *pk_rsem, ID semid);
ER	ercd =	snd_msg	(ID mbxid, T_MSG *pk_msg);
ER	ercd =	isnd_ms g	(ID mbxid, T_MSG *pk_msg);
ER	ercd =	rcv_msg	(T_MSG **ppk_msg, ID mbxid);
ER	ercd =	trcv_msg	(T_MSG **ppk_msg, ID mbxid, TMO tmout);
ER	ercd =	prcv_ms g	(T_MSG **ppk_msg, ID mbxid);
ER	ercd =	ref_mbx	(T_RMBX *pk_rmbx, ID mbxid);

Interrupt Management Functions

	void	ret_int	();
ER	ercd =	loc_cpu	();
ER	ercd =	unl_cpu	();

Memorypool Management Functions

```
ER  ercd =  pget_blf   (VP *p_blf, ID mpfid);
ER  ercd =  rel_blf    (ID mpfid, VP blf);
ER  ercd =  ref_mpf    (T_RMPF *pk_rmpf, ID mpfid);
ER  ercd =  pget_blk   (VP *p_blk, ID mplid);
ER  ercd =  rel_blk    (ID mplid, VP blk);
ER  ercd =  ref_mpl    (T_RMPL *pk_rmpl, ID mplid);
```

TIme Management Functions

```
ER  ercd =  set_tim    (SYSTIME *pk_tim);
ER  ercd =  get_tim    (SYSTIME *pk_tim);
ER  ercd =  dly_tsk    (DLYTIME dlytim);
ER  ercd =  act_cyc    (HNO cycno, UINT cycact);
ER  ercd =  ref_cyc    (T_RCYC *pk_rcyc, HNO cycno);
ER  ercd =  ref_alm    (T_RALM *pk_ralm, HNO almno);
```

Version Management Function

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

Implementation-Dependent System Call

```
ER  ercd =  vrst_msg   (ID mbxid);
ER  ercd =  vrst_blf   (ID mpfid);
ER  ercd =  vrst_blk   ();
```

3.5. Data Type

```
typedef   signed char       B;          /* Signed 8-bit integer */
typedef   signed short      H;          /* Signed 16-bit integer */
typedef   signed 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   signed char       VB          /* Unpredicatable data, signed (8-bit size) */
typedef   signed short      VH;         /* Unpredicatable data, signed (16-bit size) */
typedef   signed long       VW;         /* Unpredicatable data, signed (32-bit size) */
typedef   void far          *VP;        /* Pointer to Unpredicatable data */
typedef   void              (*FP)();    /* Start address of program general */
typedef   H                 INT         /* Signed 16-bit integer */
typedef   UH                UINT;       /* Unsigned 16-bit integer */
typedef   H                 ID;         /* ID number of object */
typedef   H                 PRI;        /* Task priority */
typedef   H                 TMO;        /* Timeout */
typedef   H                 HNO;        /* ID number of handler */
typedef   H                 ER;         /* Error code */
typedef   H                 DLYTIME;    /* Delay time */
typedef   H                 CYCTIME;    /* Interval of cyclic handler starts*/
typedef   H                 BOOL_ID;    /* Boolean value or ID number */
typedef   void near * far   PT_MSG;     /* 16-bit message data for mail box */
typedef   void far * far    PT_MSG;     /* 32-bit message data for mail box */
```

3.6. Common Constants and Packet Format of Structure

```
---- Common ----
NADR  -1       /* Invalid address and pointer value */
TRUE  1        /* True */
FALSE 0        /* False */

---- Related to Task management ----
TSK_SELF       0        /* Own task specification */
TPRI_RUN       0        /* Specifies the highest priority then under execution */
taskstat:
     TTS_RUN H'01     /* RUN */
     TTS_RDY H'02     /* READY */
     TTS_WAI H'04     /* WAIT */
     TTS_SUS H'08     /* SUSPEND */
     TTS_WAS H'0C     /* WAIT-SUSPEND */
     TTS_DMT H'10     /* DORMANT */
tskwait:
     TTW_SLP H'0001  /* Waiting with slp_tsk,tslp_tsk */
     TTW_DLY H'0002  /* Waiting with dly_tsk */
     TTW_FLG H'0010  /* Waiting with wai_flg,twai_flg */
     TTW_SEM H'0020  /* Waiting with wai_sem,twai_sem */
     TTW_MBX H'0040  /* Waiting with rcv_msg,trcv_msg */
typedef struct t_rtsk {
     VP      exinf;          /* Extended information */
     PRI     tskpri; /* Current task priority level */
     UINT    tskstat;        /* Task status */
     UINT    tskwait;        /* Wait request */
     ID      wid;            /* Wait object ID */
} T_RTSK;

---- Related to Eventflag ----
wfmod:
     TWF_ANDW        H'0000  /* AND wait */
     TWF_ORW H'0002  /* OR wait */
     TWF_CLR H'0001  /* Clear specification */

typedef struct t_rflg {
     VP              exinf;          /* Extended information */
     BOOL_ID wtsk;           /* Waiting task information */
     UINT            flgptn; /* Bit pattern of EventFlag */
} T_RFLG;
```

```
---- Related to Semaphore ----
typedef struct t_rsem {
     VP             exinf;  /* Extended information */
     BOOL_ID wtsk;   /* Waiting task information */
     INT            semcnt; /* Current semaphore count */
} T_RSEM;

---- Related to Mailbox ----
typedef struct t_rmbx {
     VP             exinf;  /* Extended information */
     BOOL_ID wtsk;   /* Waiting task information */
     PT_MSG         pk_msg; /* Starting address of next received message packet
*/
     INT            msgcnt; /* The number of messages */
} T_RMBX;

---- Related to Fixed-size Memorypool ----
typedef struct t_rmpf {
     VP             exinf;  /* Extended information */
     BOOL_ID wtsk;   /* indicates whether or not there are waiting tasks */
     INT            frbcnt; /* free block count */
     INT            blksz;  /* fixed-size memory block size */
} T_RMPF;

---- Related to Variable-size Memorypool ----
typedef struct t_rmpl {
     VP             exinf;  /* Extended information */
     BOOL_ID wtsk;   /* indicates whether or not there is a task waiting */
     INT            frsz;   /* total size of free memory */
     INT            maxsz;  /* size of largest contiguous memory */
} T_RMPL;

---- Related to Time management ----
typedef struct t_systime{
     H       utime;  /* 16 high-order bits */
     UH      mtime;  /* 16 mid-order bits */
     UH      ltime;  /* 16 low-order bits */
} SYSTIME, ALMTIME;
cycact:
     TCY_OFF H'0000  /* Cyclic handler is not active */
     TCY_ON          H'0001  /* Cyclic handler is activated */
     TCY_INI H'0002  /* Cyclic counter is initialized */
```

```
---- Related to System manegement ----
typedef struct t_ver {
     UH      maker;  /* Maker */
     UH      id;     /* Type number */
     UH      spver;  /* Specification version */
     UH      prver;  /* Product version */
     UH      prno[4];        /* Product management information */
     UH      cpu;    /* CPU information */
     UH      var;    /* Variation discriptor */
} T_VER;
```

Index

A

act_cyc.. 133
alarm handler.. 16
  reference.. 137
AND wait.. 74

B

bit pattern to be waited for 73

C

can_wup 65
chg_pri 31
Clear specification.. 74
clr_flg 71
count value of a semaphore............................ 83
CPU information... 140
cyclic handler
  activation status 133
cyclic handler 16
  active state... 135
  reference.. 135

D

Delay time.. 131
dis_dsp 27
dispatch 27, 29, 111
dly_tsk... 131

E

ena_dsp 29
eventflag
  clear...71
  get 79
  set...67, 69
  wait 73, 76
eventflag status
  reference 81
ext_tsk 23

F

fixed-size memory block
  get 115
  release.. 117
fixed-size memory block
  release all 144
fixed-size memorypool
  reference 119
Format number...140

G

get_tid...44
get_tim..129
get_ver 139

I

ichg_pri...33
interrupt handler 15, 16
irel_wai 42
irot_rdq 38
irsm_tsk 55
iset_flg 69
isig_sem 85
isnd_msg 98
ista_tsk 21
isus_tsk 51
iwup_tsk 63

L

loc_cpu ... 111

M

mailbox
clear ... 142
reference ... 107
message
receiving ... 100, 103, 105
send ... 95, 98
message queue ... 96, 101, 104, 105

O

OR wait ... 74
OS interrupt disable level ... 111
OS-dependent external interrupts ... 111

P

pget_blf ... 115
pget_blk ... 121
prcv_msg ... 105
preempted ... 111
preq_sem ... 91
priority ... 31
Product control information ... 140
Product version ... 140

R

rcv_msg ... 100
ready queue ... 35
Reason for wait ... 46
ref_alm ... 137
ref_cyc ... 135
ref_flg ... 81
ref_mbx ... 107
ref_sem ... 93
ref_tsk ... 46
rel_blf ... 117
rel_blk ... 123
rel_wai ... 40
ret_int ... 109
rot_rdq ... 35
round robin scheduling ... 38
rsm_tsk ... 53

S

scheduler ... 29
semaphore
Obtains one resource ... 87, 89, 91
reference ... 93
semaphore
Returns resource ... 83, 85
set_flg ... 67
set_tim ... 127
sig_sem ... 83
slp_tsk ... 57
snd_msg ... 95
Specification version ... 140
sta_tsk ... 18
Stack Size ... 7
sus_tsk ... 49
SUSPEND ... 49
system clock ... 127, 129
system stack ... 8, 9
System Stack ... 13

T

Task start code ... 18
Task status ... 46
ter_tsk ... 25
Timeout value ... 59
TMO_FEVR ... 60, 77, 90, 104
TMO_POL ... 77, 90
TMO_POL(... 104
TPRI_RUN ... 36
trcv_msg ... 103
TSK_SELF ... 32
tskid ... 32
tslp_tsk ... 59
twai_flg ... 76
twai_sem ... 89
TWF_ANDW ... 74
TWF_CLR ... 74
TWF_ORW ... 74

U

user stack ... 8, 9
User Stack ... 11

V

variable-size memory block
get ... 121
release ... 123
variable-size memory block
release all ... 146
variable-size memorypool
reference ... 125
Variation descriptor ... 140
version number ... 139
vras_fex ... 146
vrst_blf ... 144
vrst_msg ... 142

W

wai_flg ... 73
wai_sem ... 87
WAIT ... 57, 65
Wait mode ... 73, 79
Wait object ID ... 46
WAIT-SUSPEND ... 53, 57, 65
wakeup request count ... 62
wup_tsk ... 61

M3T-MR308 V.1.20 Reference Manual

Rev. 1.00
September 16, 2003
REJ10J0098-0100Z

COPYRIGHT ©2003 RENESAS TECHNOLOGY CORPORATION
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED

M3T-MR308 V.1.20
Reference Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J0098-0100Z